Computation of Fluid Flow

A dissertation submitted to
the University of Manchester
for the degree of Master of Science
in the Faculty of Technology

by

Kittisak Tiyapan

University of Manchester Institute of Science and Technology
Control Systems Centre
Department of Electrical and Electronics Engineering

Manchester

Supervisors : Prof. G. A. Davies and
Prof. D. J. Bell

February 2010

© Copyright 1995 M.Sc.

by

Kittisak Tiyapan

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other University or other institution of learning.

Kittisak Tiyapan

To my parents

for their love and encouragement

Abstract

Recently a lot of research has been carried out into developing models to represent the structure of porous media. The objects of this are to develop from these models procedures which can be used to predict the flux of permeate through porous media as a result of the driving force such as pressure. Additionally in many practical applications the permeate may not be a simple single phase fluid, but may contain in suspension solid particles or a second immiscible fluid phase. In a percolation process, as for example in the chemical engineering context of a filtration membrane, the concentration of the heterogeneous mixture in the porous material is not uniform. Depending on the relative dimensions of the pores and the particles, particles will concentrate at points within the membrane porous material. This will give rise to a variation in permeate flow and can well in the limit give rise to blockage so that no further permeation is possible. This type of problem has significant implications in chemical process technology and efforts to understand the mechanism of precipitation of blockage are being made. This work forms part of this programme. The blockage mechanism and its influence on permeation, is considered using percolation theory. In this thesis a model to represent the structure of the material is based on the Voronoi tessellation in two and three space. The critical probability of lattices in two or three space are computed. Calculations are made for both site and bond percolation. There are data in the literature for critical probability on two dimensional Voronoi lattices for site percolation and the data obtained in this work is compared with published values. In addition to check the computation algorithms developed in this work, calculations are made on three regular two dimensional lattices for which both site and bond percolation data is available. The comparisons of the calculations with the published data for these cases are good. The present work provides data of the critical probability for bond percolation on a two dimensional and

three dimensional Voronoi lattice. It is expected this can be used for defining conditions where blinding may become an important issue as in, for example, micro and ultrafiltration. Further work needs to be done to apply the results of the work and to examine this cross flow case. In the thesis other applications of percolation theory and the use of Voronoi lattices are described.

Acknowledgements

I would like to thank both **Prof. G. A. Davies** and **Prof. D. J. Bell** for supervising me through this project, and also for introducing me to the theory of percolation and the Voronoi tessellation.

I would like to thank **Dr. R. Jafferali** for letting me use his programs, namely *points.c* and *voronoi_plane.c*, and for his good advice toward the progress of this project.

For LaTeX lore, I would like to thank **Dr. L. R. Arnaut** at the Department of Electrical and Electronics Engineering.

Many thanks to the following computing support people, **Susan Simpson** at the Control Systems Centre, **Steven C. Meert** and **Steven C.-Lewis** at the Department of Chemical Engineering, and **Peter Lambert** at the UMIST computing support unit.

Encouragement from many people was essential at *critical* times. I cannot mention the name of everyone simply because I have to put some *limit* to the size of the list (lest it might *percolate*). I would like to thank **Dr. Suchin Arunsawatwong**, **Sunee Ngaoaram**, **Ng Gee Wah**, **Margaret Ireland**, and **Clare Entwistle** for encouraging me to go on, and for helping me in many ways through out the project. I must say that all that I had done was only the doing of the project, the possibility of completing it came from every person that I have mentioned.

Contents

Abstract **v**

Acknowledgements **vii**

Notation **xiv**

1 Introduction **1**

1.1 Objectives 1

1.2 Structural model of porous media 2

1.3 Tilings and Voronoi tessellation 2

1.4 Determination of membranes' performance 6

1.5 Problem of computing the J-ΔP relation 7

1.6 Possible applications 8

1.7 Percolation theory 9

1.8 On applications of site and bond percolations 12

1.9 Applications for Voronoi tessellation and percolation theory 12

1.9.1 Material science 12

1.9.2 Astronomy and cosmology 13

1.9.3 Electrical conductivity 14

1.9.4 Biological and medical applications 14

1.9.5 Other applications 14

1.10 Conclusion 15

2 Modelling and Generation of Voronoi Networks **16**

2.1 Direct approach . . . 16

2.2 Delaunay tessellation approach . . . 17

2.3 Divide-and-conquor approach . . . 17

2.4 Plane-sweep technique . . . 17

2.5 Algorithms survey . . . 18

3 Site percolation results of 2-D Voronoi networks **19**

3.1 Algorithm using next-nearest neighbours . . . 19

3.2 Algorithm using nearest neighbours . . . 27

3.3 Discussion . . . 28

4 Bond percolation results of 2-D Voronoi networks **29**

4.1 Algorithm using edge approximation . . . 30

4.2 Algorithm using a control area . . . 45

4.3 Discussion . . . 50

5 Bond percolation of some standard uniform lattices **52**

5.1 Square lattice . . . 52

5.2 Triangular lattice . . . 52

5.3 Honey comb lattice . . . 53

5.4 Kagomé lattice . . . 53

5.5 Discussion . . . 58

6 Discussion **59**

7 Conclusions **66**

7.1 Future work . . . 69

A Algorithms **70**

A.1 Algorithms used to prepare networks . . . 70

A.2 Algorithms used to find percolation . . . 70

A.2.1 Site percolation . . . 70

A.2.2 Bond percolation 72

B Programs **75**

B.1 Programs used to prepare networks 75

B.2 Programs used to find percolation 75

B.2.1 Site percolation 75

B.2.2 Bond percolation 80

C Generating uniform lattices **93**

C.1 Square lattice 93

C.2 Triangular lattice 94

C.3 Honeycomb lattice 94

C.4 Kagomé lattice 96

Bibliography **102**

List of Figures

1.1 *Examples of cellular structures (a) Human liver cell culture, (b) Simulated cell culture, (c) Cellular structure of bolsa, (d) Polyurethane open foam, (e) Symmetric cellulose acetate, (f) Surface of Death Valley* 4

1.2 *Duality of Voronoi tessellation and Delaunay triangles* 6

1.3 *Example of (a) a bond and (b) a site percolation* 10

1.4 *Honey comb lattice and Kagomé lattice* 11

3.1 *Site percolation for (a) 70 cells, no constraints, (b) 63 cells, limit* $= 0.01$ 20

3.2 *Site percolation for (a) 71 cell, limit* $= 0.02$*, (b) 67 cells, limit* $= 0.03$ 21

3.3 *Site percolation for (a) 69 cells, limit* $= 0.04$*, (b) 68 cells, limit* $= 0.05$ 21

3.4 *Site percolation for (a) 71 cells, limit* $= 0.06$*, (b) 70 cells, limit* $= 0.07$ 22

3.5 *Site percolation for (a) 157 cells, no constraints, (b) 154 cells, limit* $= 0.01$ . . . 22

3.6 *Site percolation for (a) 153 cells, limit* $= 0.02$*, (b) 148 cells, limit* $= 0.03$ 23

3.7 *Site percolation for (a) 156 cells, limit* $= 0.04$*, (b) 149 cells, limit* $= 0.05$ 23

3.8 *Site percolation for (a) 249 cells, no constraints, (b) 244 cells, limit* $= 0.01$ . . . 24

3.9 *Site percolation for (a) 237 cells, limit* $= 0.02$*, (b) 239 cells, limit* $= 0.03$ 24

3.10 *Site percolation for (a) 332 cells, no constraints, (b) 330 cells, limit* $= 0.01$ . . . 25

3.11 *Site percolation for (a) 334 cells, limit* $= 0.02$*, (b) 330 cells, limit* $= 0.03$ 25

3.12 *Site percolation for (a) 426 cells, no constraints, (b) 419 cells, limit* $= 0.01$ . . . 26

3.13 *Site percolation for (a) 416 cells, limit* $= 0.02$*, (b) 413 cells, limit* $= 0.03$ 26

3.14 *Site percolation for (a) 70 cells, no constraints, (b) 67 cells, no constraints* 27

4.1 *Bond percolation for (a) 233 bonds, no constraints, (b) 216 bonds, limit* $= 0.01$. 30

4.2 *Bond percolation for (a) 236 bonds, limit* $= 0.02$*, (b) 227 bonds, limit* $= 0.03$ . . 31

4.3 *Bond percolation for (a) 233 bonds, limit* = 0.04*, (b) 233 bonds, limit* = 0.05 . . 31
4.4 *Bond percolation for (a) 233 bonds, limit* = 0.06*, (b) 233 bonds, limit* = 0.07 . . 32
4.5 *Bond percolation for (a) 495 bonds, no constraints, (b) 497 bonds, limit* = 0.01 . 32
4.6 *Bond percolation for (a) 497 bonds, limit* = 0.02*, (b) 495 bonds, limit* = 0.03 . . 33
4.7 *Bond percolation for (a) 493 bonds, limit* = 0.04*, (b) 498 bonds, limit* = 0.05 . . 33
4.8 *Bond percolation for (a) 769 bonds, no constraints, (b) 781 bonds, limit* = 0.01 . 34
4.9 *Bond percolation for (a) 768 bonds, limit* = 0.02*, (b) 773 bonds, limit* = 0.03 . . 34
4.10 *Bond percolation for (a) 1059 bonds, no constraints, (b) 1053 bonds, limit* = 0.01 35
4.11 *Bond percolation for 1059 bonds, no constraints* 35
4.12 *Bond percolation for (a) 1066 bonds, limit* = 0.02*, (b) 1056 bonds, limit* = 0.03 . 36
4.13 *Bond percolation for (a) 1066 bonds, limit* = 0.02*, (b) 1056 bonds, limit* = 0.03 . 36
4.14 *Bond percolation for (a) 1347 bonds, no constraints, (b) 1335 bonds, limit* = 0.01 37
4.15 *Bond percolation for (a) 1341 bonds, no constraints, (b) 1335 bonds, limit* = 0.01 37
4.16 *Bond percolation for 1352 bonds, limit* = 0.02 38
4.17 *Bond percolation for (a) 1625 bonds, no constraints, (b) 1638 bonds, limit* = 0.01 38
4.18 *Bond percolation for (a) 1625 bonds, no constraints, (b) 1626 bonds, limit* = 0.02 39
4.19 *Bond percolation for (a) 1626 bonds, limit* = 0.02*, (b) 1612 bonds, limit* = 0.03 . 40
4.20 *Bond percolation for 1612 bonds, limit* = 0.03 41
4.21 *Bond percolation for (a) 1913 bonds, no constraints, (b) 1902 bonds, limit* = 0.01 41
4.22 *Bond percolation for (a) 1913 bonds, no constraints, (b) 1902 bonds, limit* = 0.01 42
4.23 *Bond percolation for (a) 1913 bonds, limit* = 0.02*, (b) 1902 bonds, limit* = 0.03 . 42
4.24 *Bond percolation for 2217 bonds, no constraints* 43
4.25 *Bond percolation for 8609 bonds, no constraints* 44
4.26 *Bond percolation for (a) 132 bonds, no constraints, (b) 110 bonds, limit* = 0.01 . 45
4.27 *Bond percolation for (a) 113 bonds, limit* = 0.02*, (b) 125 bonds, limit* = 0.03 . . 46
4.28 *Bond percolation for (a) 113 bonds, limit* = 0.04*, (b) 131 bonds, limit* = 0.05 . . 46
4.29 *Bond percolation for (a) 123 bonds, limit* = 0.06*, (b) 117 bonds, limit* = 0.07 . . 47
4.30 *Bond percolation for (a) 385 bonds, no constraints, (b) 396 bonds, limit* = 0.01 . 47
4.31 *Bond percolation for (a) 389 bonds, limit* = 0.02*, (b) 412 bonds, limit* = 0.03 . . 48
4.32 *Bond percolation for (a) 403 bonds, limit* = 0.04*, (b) 402 bonds, limit* = 0.05 . . 48

4.33 *Bond percolation for (a) 580 bonds, no constraints, (b) 615 bonds, limit* $= 0.01$. 49

4.34 *Bond percolation for (a) 616 bonds, limit* $= 0.02$*, (b) 606 bonds, limit* $= 0.03$ . . 49

5.1 *Bond percolation for square lattices of sizes 840 bonds and 1860 bonds* 53

5.2 *Bond percolation for triangular lattices of sizes 305 bonds and 2894 bonds* 54

5.3 *Bond percolation for triangular lattices of size 12266 bonds* 55

5.4 *Bond percolation for honeycomb lattices of sizes 438 bonds and 2800 bonds* 56

5.5 *Bond percolation for Kagomé lattices of sizes 1602 bonds and 4470 bonds* 56

5.6 *Bond percolation for a Kagomé lattice of size 6444 bonds* 57

6.1 $\bar{p_c^s}$*'s plotted against number of sites* 59

6.2 $\bar{p_c^b}$*'s plotted against number of bonds* 60

6.3 $\sigma^2_{p_c^s}$*'s plotted against sizes of networks* 61

6.4 $\sigma^2_{p_c^b}$*'s plotted against sizes of networks* 62

6.5 *Coordination number of sites plotted against network sizes* 62

6.6 *Coordination number of bonds plotted against network sizes* 63

6.7 p_c *vs.* $\frac{1}{z}$ *for lattices in general* 63

6.8 p_c^s*'s from prediction compared with those ones from simulations* 64

6.9 p_c^b*'s from prediction compared with those ones from simulations* 65

7.1 *Number of bonds as a function of number of sites for Voronoi tessellations* 67

7.2 p_c*'s bond problem, site problem, square, triangular, honeycomb, and Kagomé lattices* 68

C.1 *Producing a Kagomé lattice from a triangular lattice* 96

Notation

α	=	an atom of a lattice L
α^c	=	an atom of a lattice L^c
β	=	a bond of a lattice L
β^c	=	a bond of a lattice L^c
ε	=	porosity of membrane
ε_A	=	porosity of the separating surface
μ	=	the solvent viscosity
ξ	=	correlation length
τ	=	the tortuosity
$E(\cdot)$	=	the expected value
$\sigma(\cdot)$	=	the standard deviation
J	=	flux
K	=	permeability coefficient
L	=	a lattice
L^c	=	a covering lattice of L
L^d	=	a dual lattice to L
$limit$	=	a constraint on the smallest size of cells within the Voronoi tessellation [27].
P	=	pressure
ΔP	=	the pressure across a membrane
$P(p)$	=	*percolation probability*, the probability that fluid from a single source atom wets infinitely many other atoms
$P_N(p)$	=	the probability that fluid from a single source atom wets at least N other atoms

p_c	=	critical probability
p_c^b	=	critical probability for a bond percolation
p_c^s	=	critical probability for a site percolation
$p_c(b, L)$	=	critical probability for a bond problem on L
$p_c(s, L)$	=	critical probability for a site problem on L
r_p	=	the mean pore radius
Δx	=	membrane thickness
T or tr	=	triangular lattice
S or sq	=	square lattice
HC or hc	=	honey comb lattice
Ka	=	Kagomé lattice

Chapter 1

Introduction

The main aim of this chapter is to give a brief introduction to subjects which are relevant to the work described in this dissertation.

1.1 Objectives

Flow of fluid through random networks can be divided into two main categories.

1. Permeation of fluids through porous media [16, 40], which can be modelled by using a bond percolation.

2. Permeation through filtration membranes, which can be modelled by using a site percolation [25, 27, 53].

The main objectives of this dissertation are as follows:-

1. To study both the bond percolation and the site percolation.

2. To simulate both the bond percolation and the site percolation on computer.

3. To find the critical probability for the bond percolation of Voronoi networks from simulation.

1.2 Structural model of porous media

Porous media can be represented by various structural models, normally consist of 3-D networks of pores or capillaries [25]. Jackson [25] gave several models that can be used to represent porous media.

1. **regular structures based on regular tessellations.** There are 11 distinct types of two dimensional tessellations which are realisable. These includes three regular tessellations, namely equilateral triangle, square, and hexagonal tilings, together with eight semiregular tessellations [2]. In three dimensions there are five possible convex polyhedra; they are the tetrahedron, cube, octahedron, dodecahedron, and icosahedron [25].

2. **parallel pore model** is a simplified model where a membrane is considered as an array of parallel pores traversing from one side of the membrane to another [25].

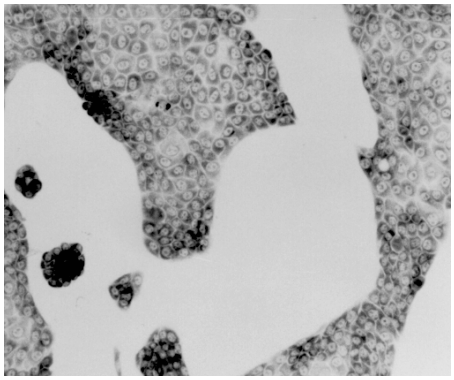
3. **packed sphere model**, an array of packed spheres, is mentioned also by Jackson [25].

4. **random model**, a good introduction of which was given by Shante and Kirkpatrick [42]. Several irregular tessellations were mentioned by Jackson [25], for example Poisson line, Poisson planar, Voronoi, and Delaunay tessellations (see also [27]).

1.3 Tilings and Voronoi tessellation

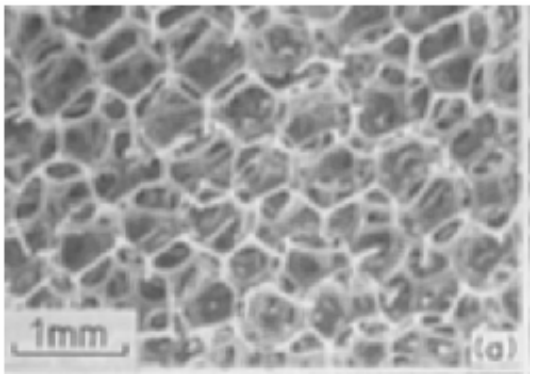
The study of the Voronoi tessellation has become essential as it has been used to describe closely many natural structures of various scales [13]. Figure 1.1 shows some examples of cellular structures. A culture of human liver cells shown in Figure 1.1*(a)* can be modelled by a two dimensional Voronoi tessellation. Simulated cell culture can be seen in Figure 1.1*(b)*, for which an analysis of the cluster patterns showed close agreement with actual cell cultures. Figure 1.1*(c)* shows a biological structure of balsa which can be represented by a 3-D Voronoi tessellation. Figure 1.1*(d)* is a synthetic material, polyurethane foam, which can also be represented by a 3-D Voronoi tessellation. The symmetric cellulose acetate shown in Figure 1.1*(e)* can be modelled by using a Voronoi tessellation [25, 27, 13], it is used as a membrane in separation process and is essentially a site problem. The surface of the Death Valley shown in Figure 1.1*(f)* filled with

cracks due to dehydration and high temperature, can be represented by a bond problem on a 2 dimensional Voronoi network [33].

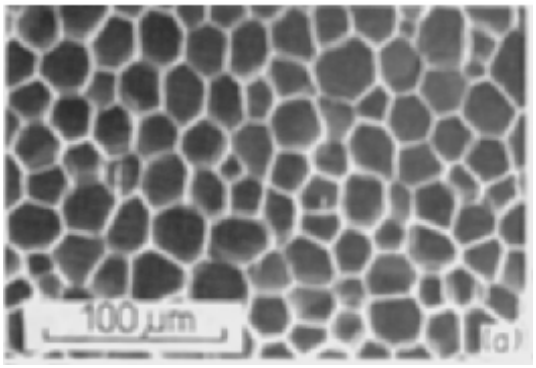


(a) Human liver cell culture

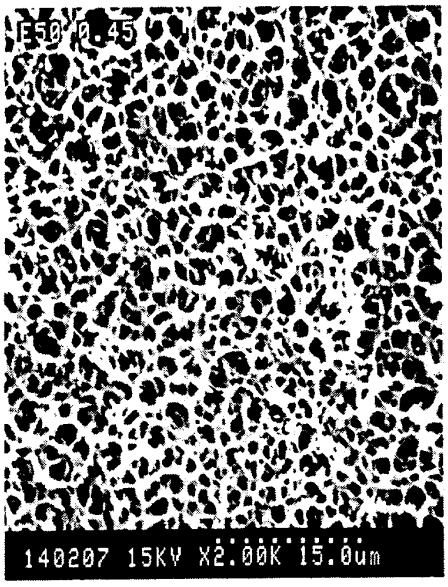
(b) Simulated cell culture



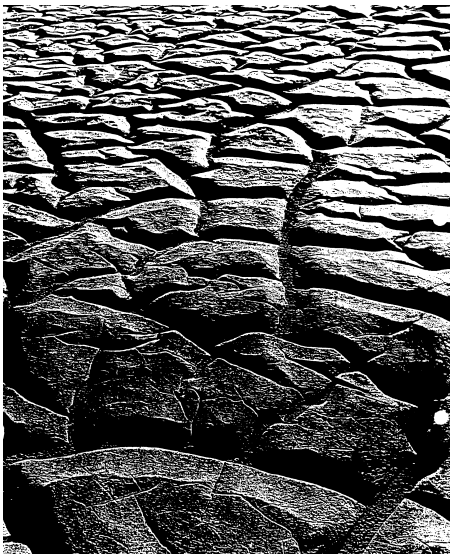
(c) Cellular structure of balsa



(d) Polyurethane open foam



(e) Symmetric cellulose acetate



(f) Surface of Death Valley

Figure 1.1: *Examples of cellular structures (a) Human liver cell culture, (b) Simulated cell culture, (c) Cellular structure of bolsa, (d) Polyurethane open foam, (e) Symmetric cellulose acetate, (f) Surface of Death Valley*

The reason that the Voronoi tessellation has been used to describe and predict structures in nature is because the statistics obtained from the Voronoi tessellation when compared to those measurements made from physical structures (for example by Scanning Electro Micrograph of membranes [27, 13] or by pencil beam survey of the universe [48]) shows very good agreement. Several statistical values are necessary in order to compare the tessellation models with those natural structures. To mention but a few of the important network statistics are, $E(x)$ (the expected value of x) and $\sigma(x)$ (the standard deviation of x), where x can represent the number of sides, the number of edges, the coordination number, the area of face, the perimeter of face, the volume of cell, etc. More about the importance of network statistics in the study of the Voronoi tessellation can be found in literature [27, 25].

A tessellation is an aggregate of cells that cover the space without overlapping. A Voronoi polygon is also known as a Dirichlet polygon, a Wigner-Seitz polygon, a Theissen polygon, a Blum's transform , an S polygon, a cell model, a plant polygon, Wirkungsbereich, etc. Early work on it was mainly in crystallography though, interestingly enough, a spider web in equilibrium state is a 2-D Voronoi diagram [3]. Though there has been more work done on 2-D than on 3-D cases, references for the 3-D Voronoi networks are not hard to find [46, 25, 27].

Definition 1.1 *Let Φ be a distribution of a countable set of nuclei $\{x_i\}$ in $\mathbb{R}^d$, and let $x_1, x_2, x_3, \ldots$ be the coordinates of the nuclei. Then, the* Voronoi region *is*

$$\Pi_i = \{x \mid d(x, x_i) < d(x, x_j) \quad \forall j \neq i\}$$

where $d(x, y)$ is the Euclidian distance between x and y.

Some authors define a Voronoi region differently, for example Klein gives both a concrete and a more abstract definition [30]. Lattices can be dual to one another, so that a site of one lattice will be a vertex of its twin, and vice versa. Dual to the Voronoi tessellation are Delaunay triangles, both are shown in Figure 1.2. The number of Voronoi points (nuclei) and edges are both $\mathcal{O}(n)$ where $n \in \mathbb{N}$, and the number of Delaunay triangles and edges are also $\mathcal{O}(n)$ [2]. ($g(n) = \mathcal{O}(f(n))$ if $|g(n)| \leq cf(n)$ when c is a constant and n is sufficiently large, ($g(n) = \Omega(f(n))$ if $|g(n)| \geq cf(n)$ when c is a constant and n is sufficiently large, and $g(n) = \Theta(f(n))$ if both $g(n) = \mathcal{O}(f(n))$ and $g(n) = \Omega(f(n))$)

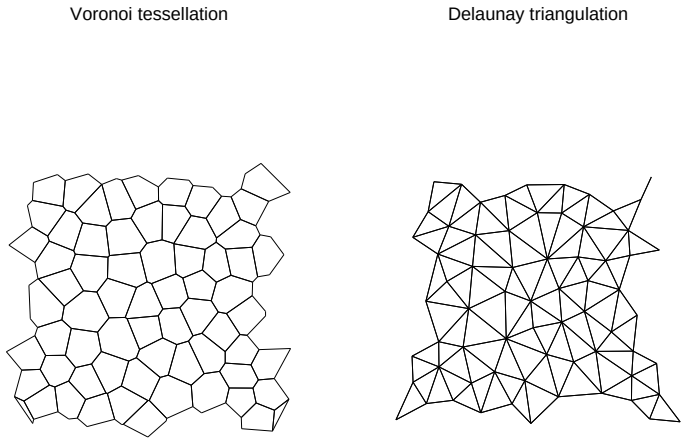


Figure 1.2: *Duality of Voronoi tessellation and Delaunay triangles*

The Voronoi tessellation shown in Figure 1.2 was created with a constraint on the smallest size of a cell. It is interesting to note that if we try to put too much constraint on the size and shape of a Voronoi cell, we end up with a lattice which approximates a honeycomb lattice. So there is a closer link between a random and a uniform lattice than one would have thought and the crystalline grid is a special case of a Voronoi tessellation [24]. There are several different variations to the Voronoi tessellation depending on how the nuclei are distributed, the limit on the distances between cells, the patterns of cells (sometimes called *mask*), etc. Many authors have contributed more variations [34, 24, 25, 27, 13]. Johnson and Mehl proposed a model (*Johnson and Mehl* model) to represent crystals growing [3], when crystals start to grow at different times, which is one of the variations of a Voronoi diagram.

For a 2-dimensional Voronoi diagram of n points the worst-case computational complexity is $\Theta(n \log n)$ [23, 3].

A *weighted Voronoi diagram* is a Voronoi diagram that has been assigned a positive *weight* $w(p)$ for each point p which describes how well p can influence its neighbours. The *weighted Voronoi diagram* contains $\Omega(n^2)$ faces, edges and vertices in the worst case [4].

1.4 Determination of membranes' performance

There are two criteria determining the performance of membranes [25].

1. selectivity

2. flow

Selectivity is described by the retention $R = \frac{c_f - c_p}{c_f}$ where c_f is the solute concentration in the feed and c_p is the solute concentration in the permeate. ($0 \leq R \leq 1$, $R = 0$ means solute and solvent have passed freely through the membrane, and $R = 1$ means complete retention)
Flow is described as flux which is defined as the volume flowing through the membrane per unit area and time. Flux is related to driving forces.

$$J = -A\frac{dX}{dx} \tag{1.1}$$

where $\frac{dX}{dx}$ is the driving force, X is either temperature, concentration, or pressure, x is the coordinate perpendicular to the transport barrier, and ΔP is the pressure across membrane thickness Δx. For flow of pure component of fluid through a membrane the Darcy's law applies.

$$J = K\Delta P \tag{1.2}$$

1.5 Problem of computing the J-ΔP relation

Jackson [25] discussed in details the problems in computing the J-ΔP relation. Many models of fluid flow through porous media use a continuum approach in which only the medium porosity and dimensions, but not the pore structures, are taken into account. This approach cannot describe the behaviour of the fluids and membranes on the microscopic scale. It cannot represent very well porous media, where internal structure changes randomly with time. One such model uses the Hagen-Poiseuille relationship (Equation 1.3), and is used for membranes consisting of straight capillaries such as in track-etched membranes.

$$J = \frac{\varepsilon_A r_p^2}{8\mu\tau} \cdot \frac{\Delta P}{\Delta x} \tag{1.3}$$

Another equation, the Kozeny-Carman equation (Equation 1.4), is used when the structure is nodular, that is sintered membranes.

$$J = \frac{\varepsilon^3}{K\mu S^2(1-\varepsilon)^2} \cdot \frac{\Delta P}{\Delta x} \tag{1.4}$$

where S is the surface area of the packing material [25].
In both Equations 1.3 and 1.4 the J-ΔP relationship depends on the porosity, pore radius of the membrane, viscosity, and other physical data of the permeant, while treating the membrane as a

black box. But the structure of the membrane (its structure and material) determines selectivity and flux, and must not be neglected. Jackson [25] gave stochastic models used to represent membranes which are based on Voronoi tessellations and take into account the internal structure of the networks. He gave rigorous descriptions of flow of fluid past networks of cylinders. His is essentially a site approach where cylinders are edges in 3-D Voronoi networks, and fluids flow past the exterior of the cylinders. In contrast, applying Voronoi networks to sintered membranes, a bond approach is used in this dissertation where fluids flow, and blockages occur, inside the cylinders.

1.6 Possible applications

A list of applications for the work in this dissertation is a long one. But we can roughly summarize into two groups according to the two main categories of networks.

- **Bond percolation** [16, 40] finds application in hydrology (for example movement of water in dam structure, intrusion of sea water in coastal areas, filter beds, etc.), petroleum engineering (for example petroleum and natural gas production, exploration, logging, etc.), chemical engineering (for example heterogeneous catalysis, flow through packed beds, gel permeation chromatography, porous polymer films used in separation processes, biological membranes, inorganic membranes, etc.), medicine and biomedical engineering (for example biological membranes, biological filters, flow of blood and other body fluids, electro-osmosis, etc.), and electrochemical engineering (for example porous electrodes, permeable and semipermeable diaphragms for electrolytic cells, etc.).

- **Site percolation** [25, 27, 53, 31] finds applications in permeation through filtration membranes [5], microfiltrations [27], sieve blinding [53]. Lim [31] investigated the simulation of membrane fouling, of the Anotec (ANOPORE) microfiltration membrane and in the form of pore clogging. He also investigated the effect of back-flushing and crossflow microfiltration. Close agreement with experiments was obtained. Readers who are interested in this area of application can find more information from the literature [32, 36, 37, 43].

1.7 Percolation theory

The study of percolation theory is justified by the fact that it can explain very well the phenomena called *phase transition*, which is the prevalent feature in many branches of science (for example filtration [5], ferromagnetic phase transition, solid-liquid phase transition, etc.). The theory can be used to describe and predict the critical behaviour of various structures because there exists a unique value of *transition threshold* called *critical probability* for each structure.

The *percolation process* can be called a dual problem to the *diffusion process*. Percolation theory describes the flow of fluid in a *random medium* while a diffusion process describes the randomness of fluid flow [5]. The term *percolation process* was first introduced by Broadbent and Hammersley [8]. Since then, it has been studied and developed extensively by people in many different professions, including engineers, physicists, physicians, cosmologists and mathematicians. This is probably due to the fact that the theory gives a model that closely resembles many phenomena in nature, some of which cannot be explained by other existing models. The research activities on percolation theory can be classified roughly into two areas, that is percolation on a medium of uniform structure and percolation on a random network.

In their paper, Hammersley and Broadbent have introduced a few important terms, for example, *atoms*, *bonds* and *self-avoiding walks*. *Atoms* are sometimes known as *sites* while *bonds* are sometime known as *edges*.

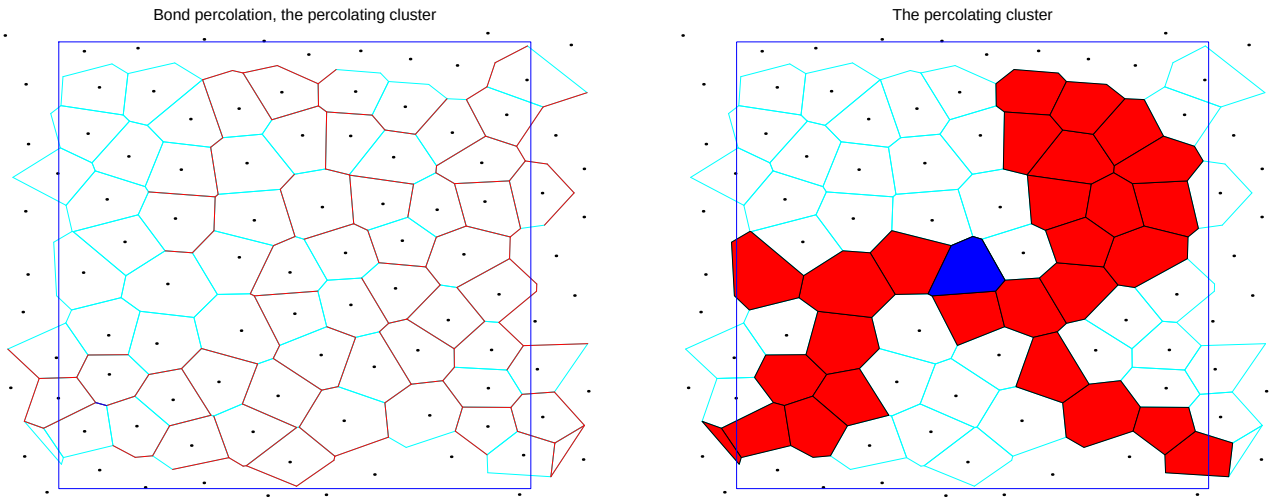
1.7.0.1 Bond-problems and site-problems

A bond-problem of any lattice L can be translated into a site-problem on L^c, its covering lattice constructed [42] by the following procedure.

1. Replacing each β_i by α_i^c.
2. Creating β_i^c by the rule that $\forall \alpha_i^c, \alpha_j^c \in L^c$, α_i^c and α_j^c are connected by a bond of L^c if and only if their corresponding β_i *and* $\beta_j \in L$ have a common terminal atom of L.

Figure 1.3 shows examples of both the bond and the site percolation of a Voronoi tessellation. An example of a $L - L^c$ pair are the honeycomb and the Kagomé lattice (Figure 1.4). Then,

$$p_c(b, L) = p_c(s, L^c) \tag{1.5}$$



(a) Example of a bond percolation *(b) Example of a site percolation*

Figure 1.3: *Example of (a) a bond and (b) a site percolation*

A *dual lattice* L^D has each of its bonds intersecting a bond of L once and only once, and vice versa.

$$p_c(b, L) + p_c(b, L^D) \geq 1 \tag{1.6}$$

A lattice L^* is said to *contain* L if L can be obtained from L^* by deleting some of the bonds of L^*. Then,

$$p_c(s, L^*) \leq p_c(s, L) \tag{1.7}$$

As an example, a triangular lattice contains a square lattice and a square lattice contains a honeycomb lattice (see [42]), so that $p_c(s, T) \leq p_c(s, S) \leq p_c(s, HC)$

Bonds connect sites together. Two *neighbouring sites* are connected by one bond and two *neighbouring bonds* shares a common site. Neighbourhood connects bonds(sites) together, forming clusters. If p^o is the probability of a bond(site) being occupied, then when $p \ll 1$ the occupied bonds(sites) are sparse and isolated. In such case the set of all clusters consists mainly of those with size one. But if we consider vacant bonds(sites) instead of occupied ones, we will see that there is hardly any cluster of vacant bonds(sites) of size one and there is a cluster of vacant bonds(sites) which percolates through the network. Here $p^v = 1 - p^o \approx 1$. Thus the relationship between the cluster size and p is that, as p increases, so does the cluster sizes. Somewhere between the two values of p there exists a *critical probability* p_c where the network changes from

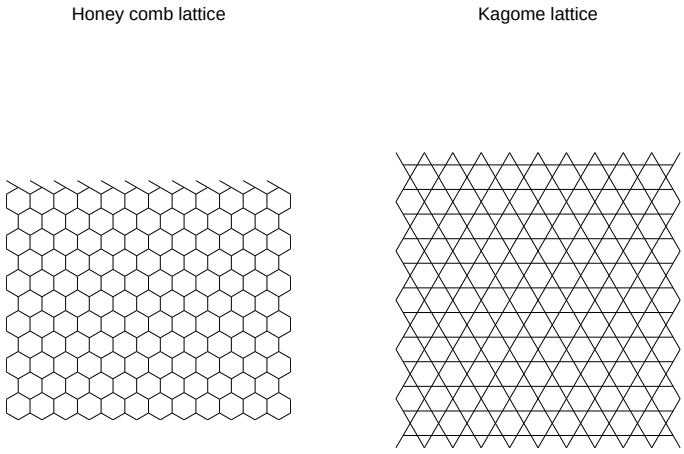


Figure 1.4: *Honey comb lattice and Kagomé lattice*

being predominantly vacant bonds(sites) into being predominantly occupied bonds(sites), that is a *phase transition* has occured. Also, at the *critical probability* the first percolating cluster, an infinite cluster, occurs. This idea of an infinite cluster can be deduced from the simulation results which show that, no matter what the sizes of the networks used in doing the simulation are, percolation always occurs at p_c.

1.7.0.2 Critical probability

At and above the critical probability a percolating cluster, a cluster that spans infinite length, occurs. Because of the irregular structure of the percolating cluster, it is necessary to know the *fractal dimension* of the clusters in order to analyse the geometrical variation of clusters at the critical point [5, 12, 35]. One important formula is the area-perimeter relation, $perimeter = area^D, D \in \mathbb{R}$ [5].

Here is some mathematical background in the study of the critical probability,

$$P(p) = \lim_{N\to\infty} P_N(p) \tag{1.8}$$

$$p_c = \sup\{p,\ such\ that\ P(p) = 0\} \tag{1.9}$$

$$P_N^s(p) \leq P_N^b(p) \tag{1.10}$$

There are several ways one can find the value of p_c, for example by Monte Carlo method, by the idea of *matching lattices* [42], by series method [10] and by simulation as will be used here.

There are many references which give values of p_c for uniform polygons and polyhedra [10, 42]. One interesting phenomenon to note is the appearance below p_c and disappearance above p_c of large isolated clusters [42]. This can be thought of as a transition from one phase to another, thus we may say that a phase transition occurs at p_c.

1.8 On applications of site and bond percolations

Site percolation has been successfully applied to simulate membrane and filter fouling mechanisms [25]. Jackson [25] described the flow of fluids past random networks of cylinders (based on the Voronoi tessellation). He then did stochastic simulations of membrane fouling by taking into account the interactions both between particle-particle and between particle-pore. The results obtained agreed very well with the statistics obtained from SEM as well as the results obtained from experiments using Whatman filters.

With a bond percolation model, we can think of membranes or filters, such as sintered materials, as being networks of tubes placed at edges of a Voronoi network. The fluids thus flow within the tubes, and fouling occurs by blockages of the tubes.

1.9 Applications for Voronoi tessellation and percolation theory

As mentioned earlier, many names have been given to Voronoi polygons, this is due to the fact that it has been studied and used by many people of different professions. It is called *Wigner-Seitz polygons* after Wigner and Seitz who first used it in metallurgy, *Theissen polygons* after Theissen who used it in forcasting precipitation, *Blum's transform* after Blum who used it in visual science, and *Voronoi polygons* or *Dirichlet tessellations* after Voronoi and Dirichlet who were pioneer mathematicians and introduced it. One variation of it is called a *Johnson-Mehl* model after Johnson and Mehl who used it in mineralogy. [3]

1.9.1 Material science

Voronoi tessellation, also known as Dirichlet tessellation, has been used to describe particle distributions in materials [34]. It has also been used, together with Molecular Dynamics (MD) computer simulation, to study the structural properties of mixtures. Sopek et al. studied

dynamical properties of molten mixtures of LiBr and KBr with different compositions and temperatures [44].

In the study of the magnetism of materials, p_c is the lower bound to the concentration of spins above which the ferromagnetic phase transition occurs. When $p = p_c$ the ferromagnetic Curie point occurs at zero temperature [18, 6].

An order-disorder transition is very important in physics as well as in material science, examples consider a solid-liquid transition, glass-liquid transition. It can be simulated on computer using a structure of Voronoi polygons [19, 22]. Percolation can be used to describe gelation processes [29]. In the study of gelation of polymer p_c is proportional to the minimum number of crosslinks per molecule needed to ensure gel formation [18].

1.9.2 Astronomy and cosmology

V. Trimble mentioned in his article [48] about a controversial observation in astronomy (a pencil-beam survey) which leads to a plot of numbers of galaxies as a function of redshifts showing sharp equally spaced peaks. He also mentioned about different approaches in trying to find a model that would fit the observation. One geometrical approach uses the idea that the galactic distribution resembles a cellular structure, with void playing a prominent role [50]. In this sense, the regions that are local positive energy perturbations can be considered as nuclei of a *Voronoi foam*, which is a realisation of the *Voronoi tessellation*. Material flowing out of adjacent voids will meet each other at walls which separate them (*pancakes*), edges (*filaments*) where three walls intersect, and nodes (*clusters*) where four filaments meet.

Percolation phase transition has been used to explain why in some galaxies matter is mostly in the form of gas while in others it is much in the form of stars (*stochastic self-propagating star formations*). For spiral galaxies, this can be simulated on computer using a 3-dimensional model, ie 2 space dimensions and one time dimension. Such models have been used to study the formation, and the measurement of pitch angles, of spiral arms in galaxies [39].

Applications of the Voronoi tessellation, especially to astrophysical studies, induce the necessity to study in detail the embedding statistics of the tessellation, for example correlation (λ) and anticorrelation (δ) parameters, circumference, area of cells, number of vertices per cell, length of wall, angle at vertex, distance between centre and node, distance between centre and wall,

2-point correlation functions (ϵ) [24, 51, 50]. The problem is essentially that of a directed bond percolation which is highly anisotropic, with

$$\xi_\alpha(p) \sim |p - p_c|^{-\nu_\alpha}, \alpha = \|, \perp$$

with $\|$ and $\perp$ are the timelike and spacelike directions respectively and $\nu_\| > \nu\perp$ [41].

1.9.3 Electrical conductivity

Electronic conduction of disordered materials is another area where percolation theory gains popularity. The existance of donors and acceptors in semiconductors makes it very suitable to apply percolation theory [42]. Metal-insulator transition resembles a transition in percolation theory [15]. Estimating the resistance of a macroscopic structure can be thought of as a problem of finding the resistivity of a random network of impedances, which is another important area of application for percolation theory.

1.9.4 Biological and medical applications

Many structures in Biotechnology and Biology can be closely represented by Voronoi diagrams [13]. Simulations of cell growth using the Voronoi tessellation have been carried out and investigated by Lim [31]. Jafferali (1995) used it to simulate animal cell cultures [27]. He also simulated the cell cultures confined within given surface boundaries, for example a plane, a sphere, an ellipsoid, and a torus. The bond problem has been successfully applied to the network of genes, which shows all the important characteristics including the criticallity at p_c where the number of isolated genes decreases rapidly [42].

The sand pattern formed by male mouth breader fishs, *Tilapia mossambica*, approximates a pattern of Voronoi polygons [47].

Cluster analysis of Voronoi diagrams has been used in the analysis of clusters and scotomas in eyes in order to improve an early detection of visual field damage in glaucoma [17].

1.9.5 Other applications

Many phenomena in physics are closely related to the percolation theory. Stauffer draws an analogy between the critical probability and the liquid-gas critical temperature [45]. He also

mentions applications in forest fire modelling (to predict whether a forest fire is going to burn down the whole forest) and oil fields modelling (to predict how much oil can be extracted from a well). The analogy is, in oil field modelling one measure the porosity in a sample of a few centimetres in size and *extrapolate* to the reservoir scale, in percolation theory one measure p_c from a simulation on a relatively small model, say 10,000 cells, and extrapolate to infinity (*infinite cluster*).

Voronoi diagrams can be applied to the area of computer science under the name of *associative file searching* problem (or post-office problem, given a set S of n sites in the plane, report a site closest to a given query point q) [3]. The same idea can also be applied to the study of location of voting places within districts [47]. In agriculture, a systematic study of a penetrometer (used to evaluate fruit firmness) has been done by computer simulation with a Voronoi model [38]. Other applications which use it as a tool are finite element, pattern classification in statistics, dot pattern analysis [1], motion planning in robotics, etc.

1.10 Conclusion

One possible application of the Voronoi tessellation is that of the prediction of membrane performance using simulation on Voronoi tessellation models. As an example, Jackson has shown that such simulation gave better agreement between the predictions of the pore properties and the data obtained by image analysis of SEM photomicrography, at the same time as giving better agreement between prediction of the relationship between the clear membrane flux versus pressure drops and that obtained from the experiments [25, 26]. The flux-pressure drop relation can be obtained from a Voronoi model by considering viscous flux around all cylinders (edges joining vertices) and the force upon each cylinder for a given flux. The fouling mechanism of membranes can be obtained from simulation by considering the particle-particle interaction and the particle-pore interaction of particles entering a Voronoi network.

For bond percolation the method mentioned above must be adapted to take into account the flow of fluids within cylinders instead of the flow past the exterior of cylinders.

Chapter 2

Modelling and Generation of Voronoi Networks

There are many ways to compute and generate a Voronoi diagram on computers. This chapter will introduce some of the approaches, others can be found in references cited. Very good introductions are given by many authors [3, 2].

2.1 Direct approach

This approach uses Definition 1.1 in a straight forward manner. An area is divided into $n \times n$ pixels and nuclei of the Voronoi cells are randomly placed in it. Then, for each individual pixel, distances from it to every nuclei are calculated and sorted in ascending order. Let $D_i, i \in \mathbb{N}$ be the sorted distances, so that $D_1 < D_2 < D_3 < \ldots$. Then, $\exists \epsilon \in \mathbb{R}, \epsilon \ll 1$ such that (for 2-dimensional case) the following algorithm holds.

if $|D_1 - D_2| > \epsilon$ **then**
    the pixel is within the area of a cell
else if $|D_1 - D_2| < \epsilon$ *but* $|D_1 - D_3| > \epsilon$ **then**
    the pixel is on an edge between two cells
else
    the pixel is on a vertex
end if

This approach is the easiest one to understand but the computational time required is prohibitive.

2.2 Delaunay tessellation approach

A Voronoi tessellation can be constructed from its dual, the Delaunay tessellation (see Figure 1.2) [2]. The geometric duality between the two makes it easy to construct one from the other. One possible way is to use the *incremental approach* by updating the diagram after a site has been added. It is then possible to build a larger Voronoi diagram upon an existing one. Many approaches have been made in speeding up insertions. One such approach, *randomization*, gives an optimal time performance ($\mathcal{O}(n \log n)$) and can be extended to an abstract Voronoi diagram without increase in runtime [3].

Still having some limitations from the fact that most algorithms in computational geometry are sequential in nature, efforts have been made to parallelise the construction of Voronoi diagrams. Computations have been done mostly by using the CREW PRAM model (Concurrent-Read, Exclusive-Write Parallel Random Access Machine), a synchronous parallel model where processors share a common memory that allows concurrent reads, but no two processors may simultaneously write to the same cell [11].

2.3 Divide-and-conquor approach

In the divide-and-conquor technique [30, 2] the problem is partitioned into subproblems of smaller size and are solved separately. The solutions of the parts are then merged together to form the solution of the original problem.

2.4 Plane-sweep technique

The plane-sweep technique attained the optimal $\mathcal{O}(n \log n)$ time and the optimal $\mathcal{O}(n)$ space [3]. In this technique (called sweep-line technique by some authors) the Voronoi diagram is first transformed using Fortune's transformation [3, 30]. Then a horizontal line l is moved upward across the plane and the set of *active* Voronoi edges, those edges intersecting l at each time step,

is stored. Whenever this set changes, l is halted, and the information about the active edges is updated.

2.5 Algorithms survey

In constructing a Voronoi diagram in a plane, Green and Sibson used an incremental algorithm which gives a performance of $\mathcal{O}(n^{\frac{3}{2}})$ [21]. Guibas et al gave an optimal randomized incremental algorithm with expected $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space when n is the number of nuclei [23]. Cole et al. gave a nice introduction about a parallel method for Voronoi diagram construction. Their approach, based on a parallel algorithm and the idea of divide-and-conquor, solves a Voronoi problem in $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(n \log n)$ processors [11]. An algorithm used by Fraser et al. [19], which follows the divide-and-conquor approach, runs in $\mathcal{O}(n \ln n)$ time. The *Convex Hull Insertion* algorithm proposed by Tsai [49] runs at approximately $\mathcal{O}(n)$ time.

In their paper [20], Fu and Lee gives an algorithm for generating a dynamic Voronoi diagram for moving nuclei with processing time of $\mathcal{O}(n)$ and preprocessing time of $\mathcal{O}(k^2 n^3 \log n \cdot 2^{\mathcal{O}(\alpha(n)^{5k+1})})$.

Aurenhammer and Edelsbrunner [4] gave an optimal algorithm which constructs a weighted Voronoi diagram in $\mathcal{O}(n^2)$ time.

One of the algorithms used to construct a Voronoi polyhedra was introduced by Brostow et al. [9]. Their method is dominated by an $\mathcal{O}(n^2 \log n)$ routine.

Some authors gave generalized algorithms for creating Voronoi tessellation (or Delaunay tessellation) in k-dimension. The algorithm given by Watson [52] constructs a k-dimensional Delaunay triangulation in $\mathcal{O}(n^{\lceil \frac{(2k-1)}{k} \rceil})$ time, using $\mathcal{O}(n^{\frac{k-1}{k}})$ storage. That given by Bowyer [7] constructs a k-dimensional Voronoi tessellation with computational time of $\mathcal{O}(a_k \cdot n^{1+\frac{1}{k}} + b_k \cdot n)$ where $a_k, b_k = constant$ and both depend upon k.

Algorithms used in finding neighbours and next-nearest neighbours normally run in $\mathcal{O}(n^2)$ time, which is the same as the one used here. Fraser et al. used a method in updating information of neighbours and next-nearest neighbours for a dynamic Voronoi problem which gives a $\mathcal{O}(n)$ time performance [19].

Chapter 3

Site percolation results of 2-D Voronoi networks

Site percolation on a Voronoi tessellation can be used to describe permeation of fluids through filtration membranes [25, 27, 53, 28, 5, 14]. It is very important in the study of separation processes where solid particles are removed from fluid. As an example, Voronoi tessellation was shown to be a good representation of a porous cellulose acetate membrane [27]. The two dimensional Voronoi network represents a cross section of the cellular structure of a membrane and percolation can be interpreted as the blocking of the membrane.

For the site percolation problem two different algorithms were tried for finding p_c^s. Both algorithms used are listed in Appendix 1 and the programs used are listed in Appendix 2. All the simulations in this chapter were done by MATLAB.

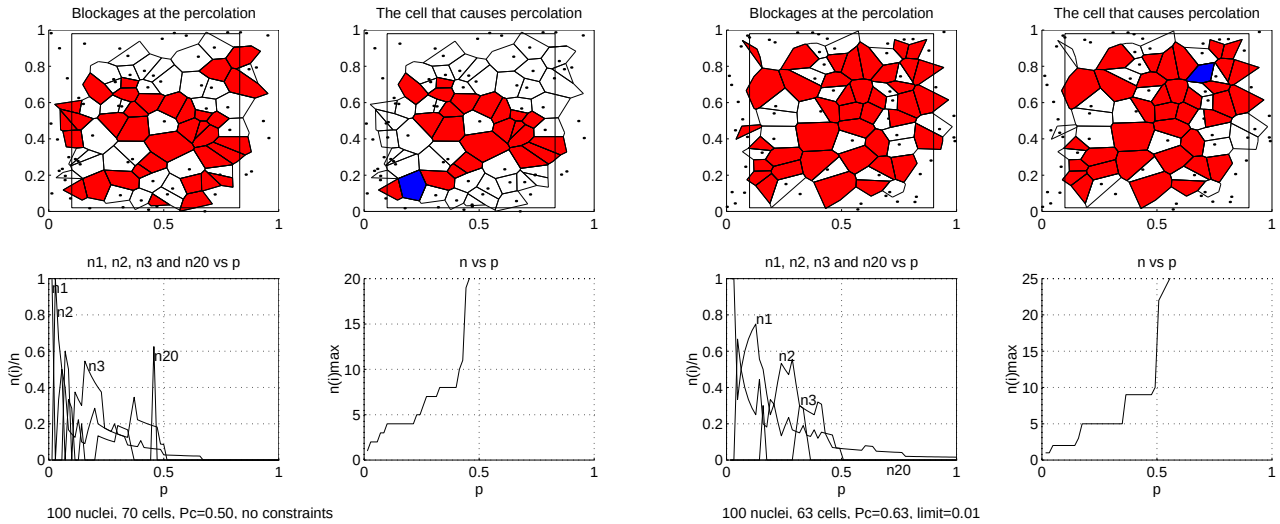
Figures 3.1 - 3.14 that follows show the results from simulations. For some of the figures,

$$\begin{aligned} n &= \textit{total number of blocked sites} \\ n(i) &= \textit{number of sites in cluster } i \\ n(i)_{max} &= \textit{size of the largest cluster} \end{aligned}$$

3.1 Algorithm using next-nearest neighbours

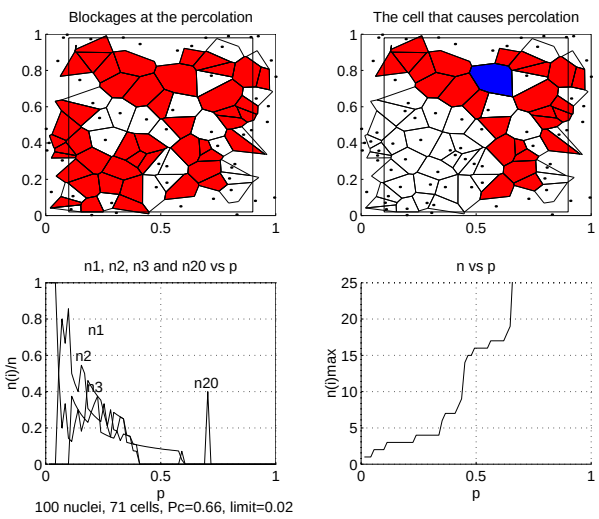
Site percolation simulations were done using Algorithm 1 which takes into account next-nearest neighbours as well as nearest neighbours. Next-nearest neighbours are cells which share only

one common vertex while neighbours are cells which share two.

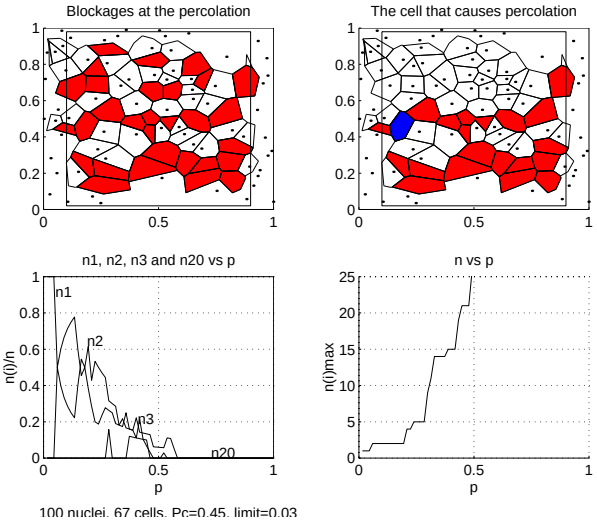


(a) 70 cells, $p_c^s = 0.50$ *(b) 63 cells, $p_c^s = 0.63$*

Figure 3.1: *Site percolation for (a) 70 cells, no constraints, (b) 63 cells, limit* $= 0.01$

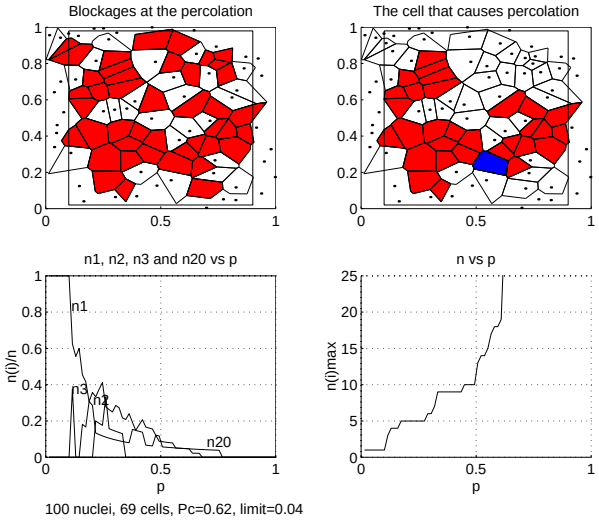


(a) 71 cells, $p_c^s = 0.66$

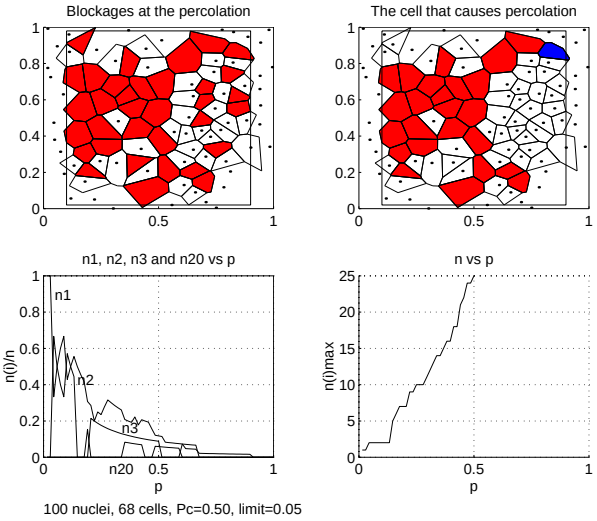


(b) 67 cells, $p_c^s = 0.45$

Figure 3.2: *Site percolation for (a) 71 cell, limit* $= 0.02$*, (b) 67 cells, limit* $= 0.03$

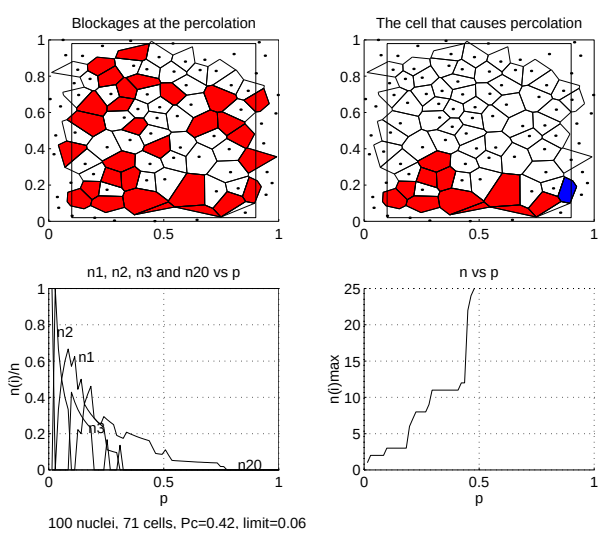


(a) 69 cells, $p_c^s = 0.62$

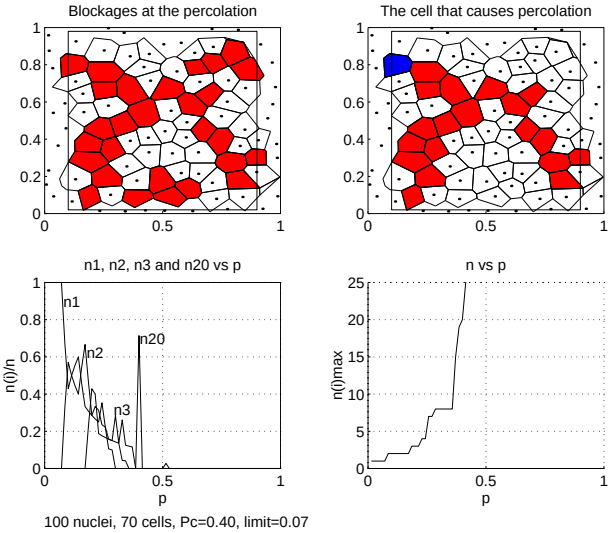


(b) 68 cells $p_c^s = 0.50$

Figure 3.3: *Site percolation for (a) 69 cells, limit* $= 0.04$*, (b) 68 cells, limit* $= 0.05$

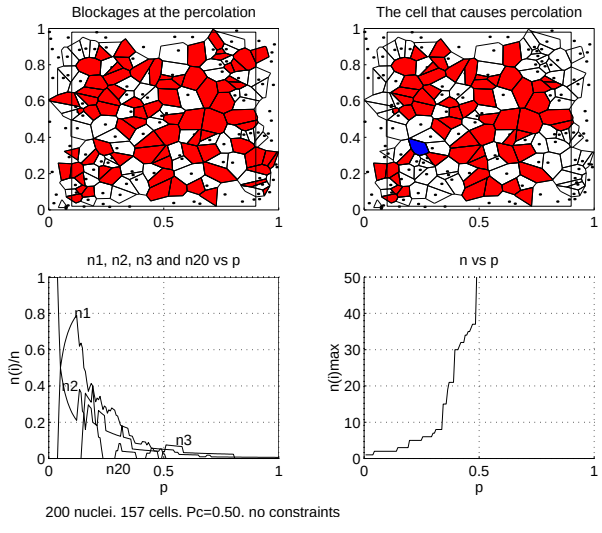


(a) 71 cells, $p_c^s = 0.42$

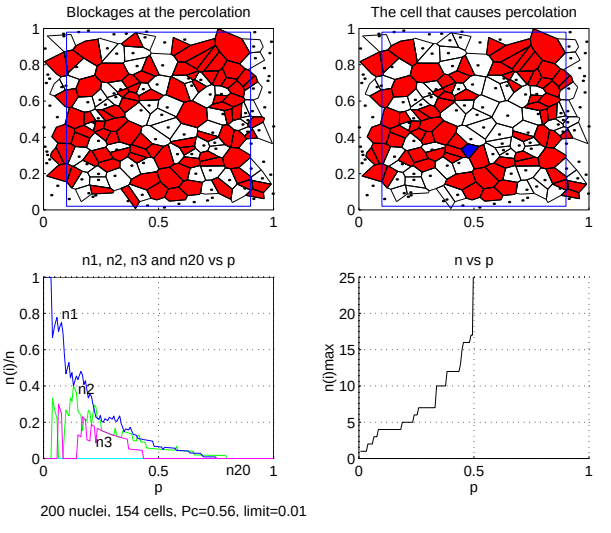


(b) 70 cells, $p_c^s = 0.40$

Figure 3.4: *Site percolation for (a) 71 cells, limit* $= 0.06$*, (b) 70 cells, limit* $= 0.07$

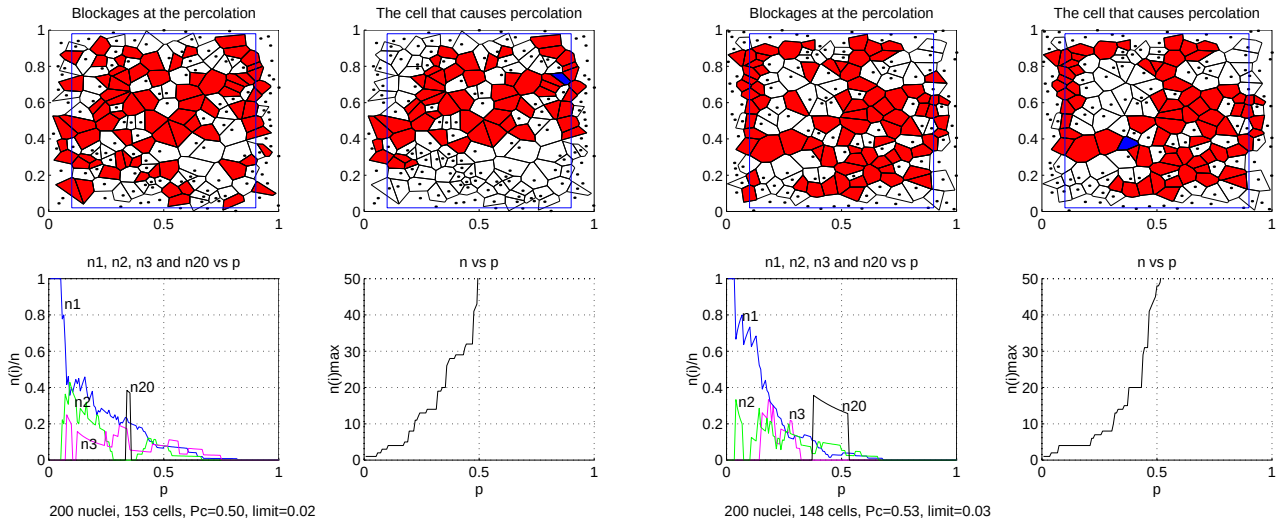


(a) 157 cells, $p_c^s = 0.50$



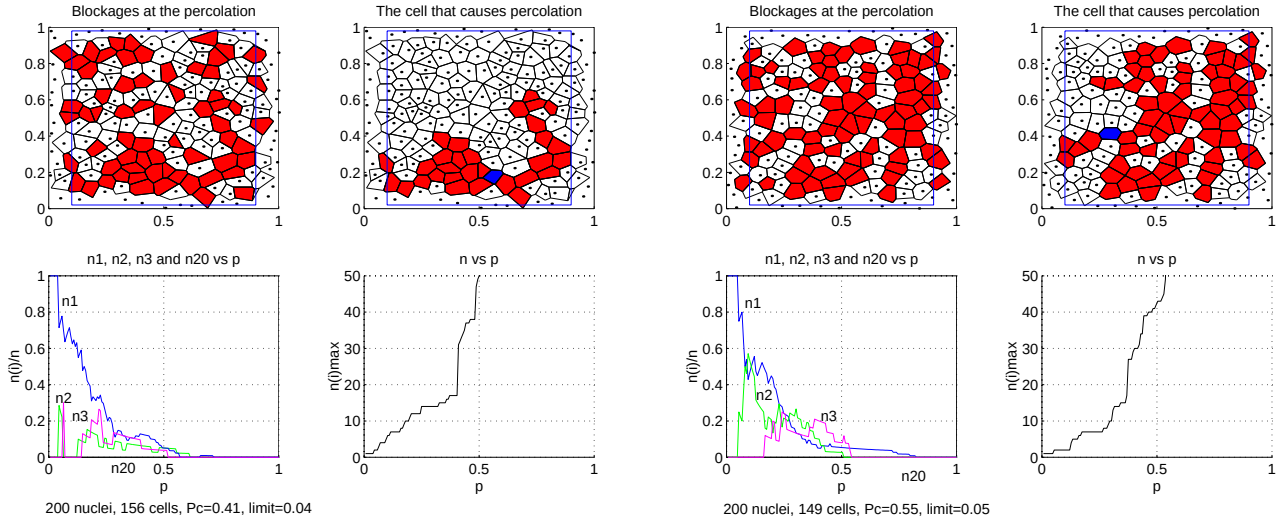
(b) 154 cells, $p_c^s = 0.56$

Figure 3.5: *Site percolation for (a) 157 cells, no constraints, (b) 154 cells, limit* $= 0.01$



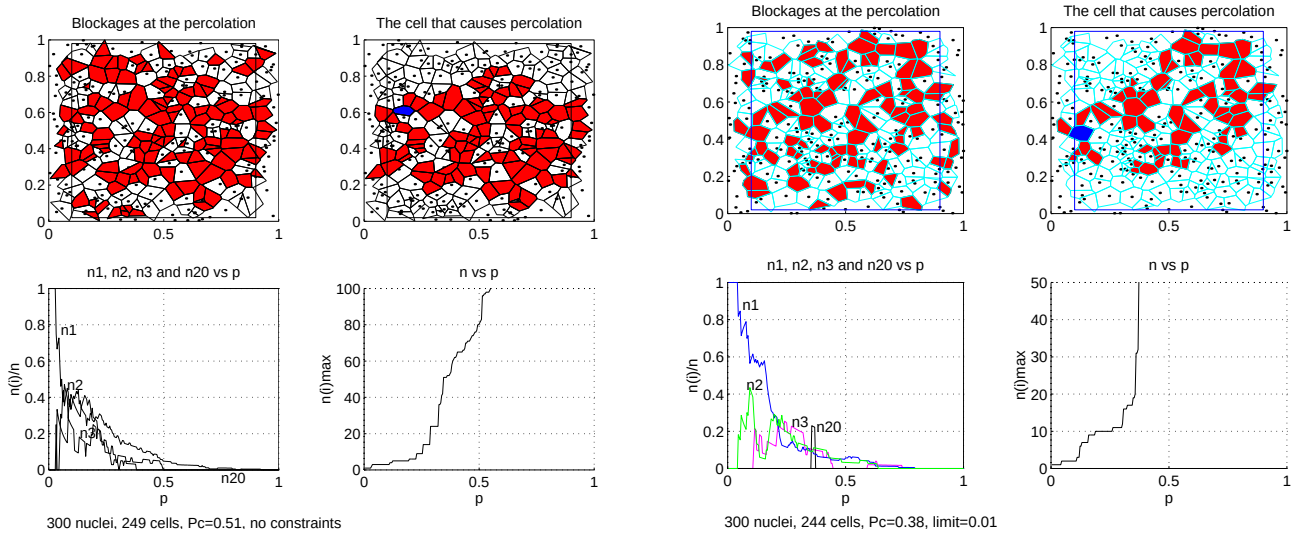
(a) 153 cells, $p_c^s = 0.50$ *(b) 148 cells,* $p_c^s = 0.53$

Figure 3.6: *Site percolation for (a) 153 cells, limit* $= 0.02$*, (b) 148 cells, limit* $= 0.03$



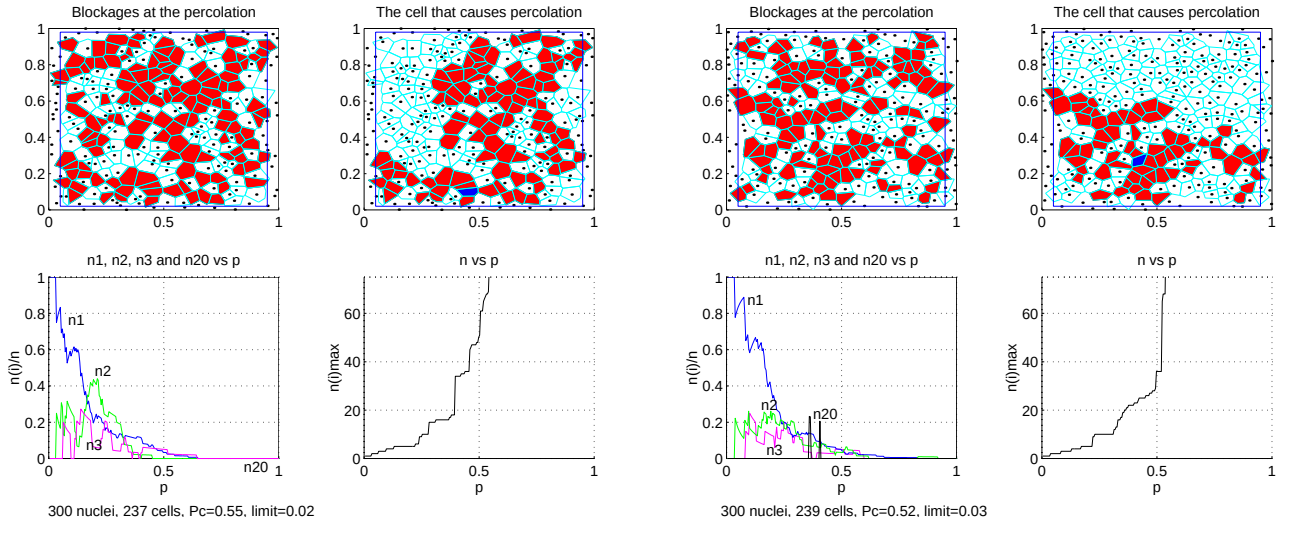
(a) 156 cells, $p_c^s = 0.41$ *(b) 149 cells,* $p_c^s = 0.55$

Figure 3.7: *Site percolation for (a) 156 cells, limit* $= 0.04$*, (b) 149 cells, limit* $= 0.05$



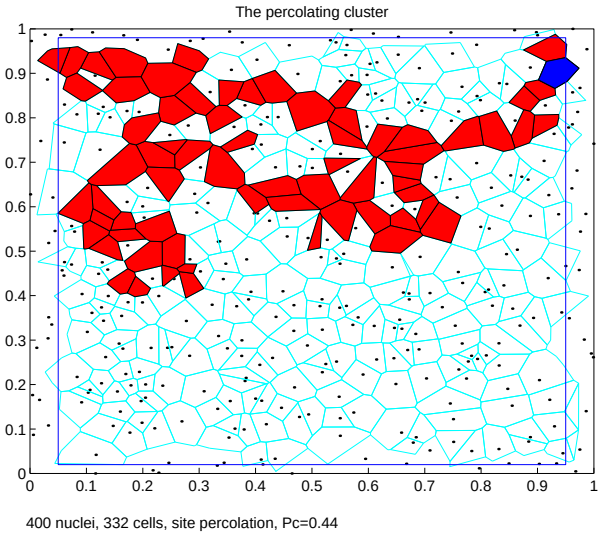
(a) 249 cells, $p_c^s = 0.51$ *(b) 244 cells,* $p_c^s = 0.38$

Figure 3.8: *Site percolation for (a) 249 cells, no constraints, (b) 244 cells, limit* $= 0.01$

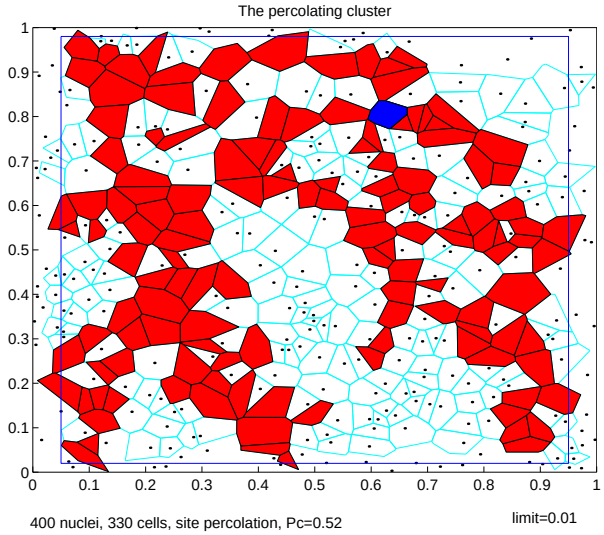


(a) 237 cells, $p_c^s = 0.55$ *(b) 239 cells,* $p_c^s = 0.52$

Figure 3.9: *Site percolation for (a) 237 cells, limit* $= 0.02$*, (b) 239 cells, limit* $= 0.03$

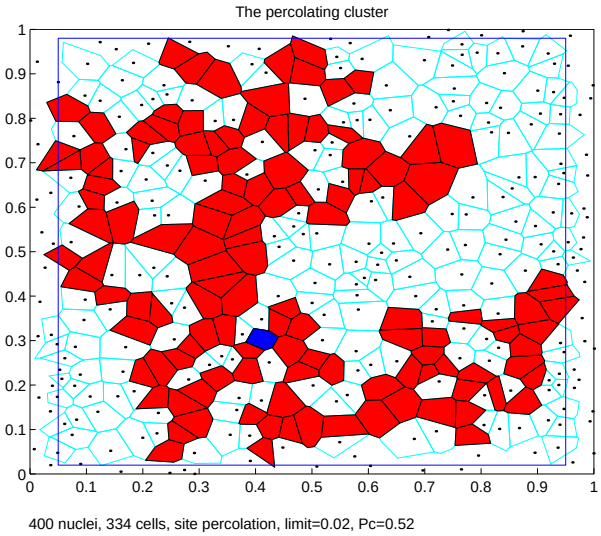


(a) 332 cells, $p_c^s = 0.44$

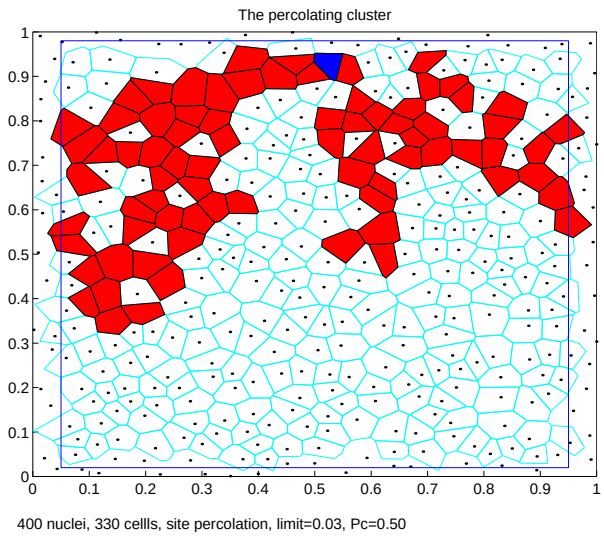


(b) 330 cells, $p_c^s = 0.52$

Figure 3.10: *Site percolation for (a) 332 cells, no constraints, (b) 330 cells, limit* $= 0.01$

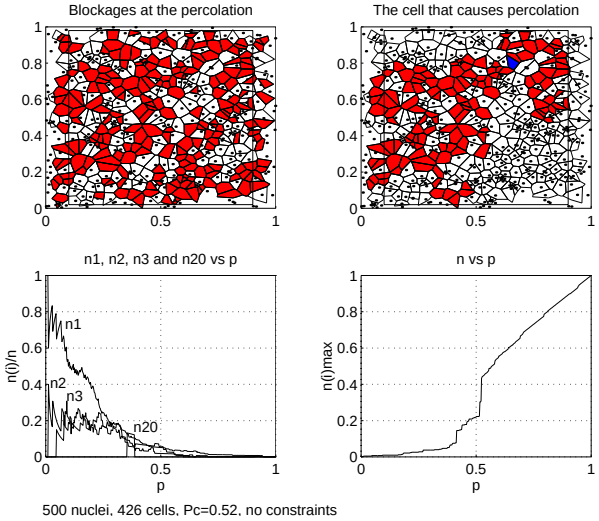


(a) 334 cells, $p_c^s = 0.52$

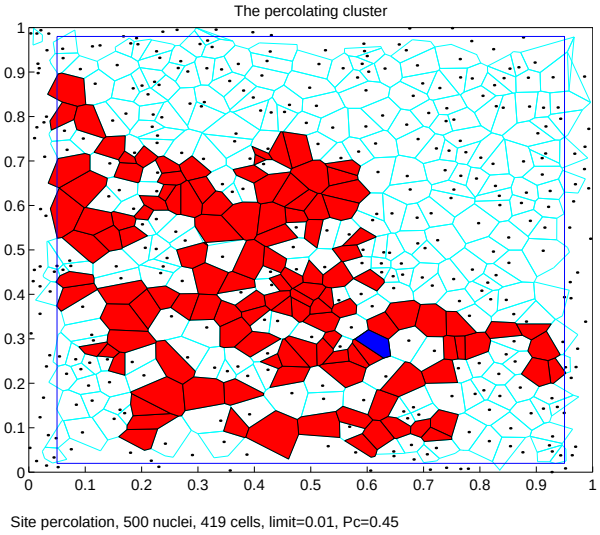


(b) 330 cells, $p_c^s = 0.50$

Figure 3.11: *Site percolation for (a) 334 cells, limit* $= 0.02$*, (b) 330 cells, limit* $= 0.03$

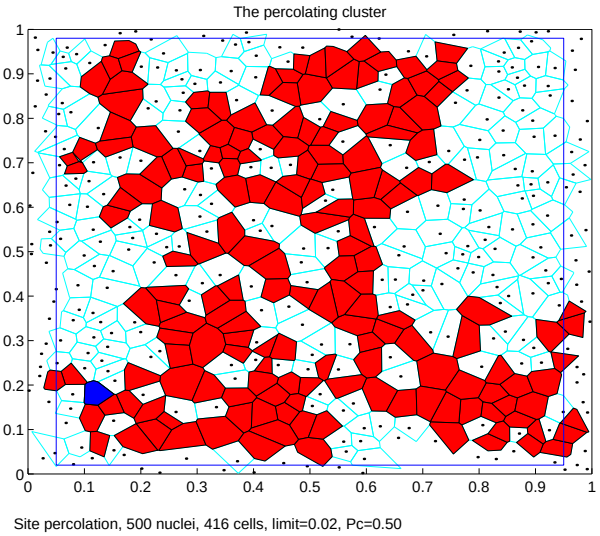


(a) 426 cells, $p_c^s = 0.52$

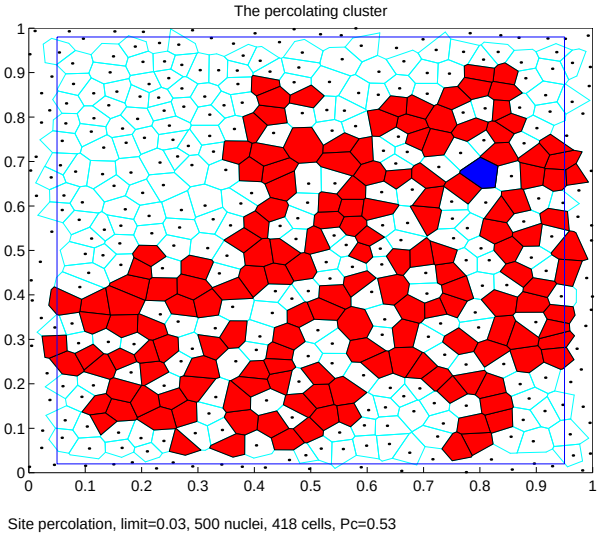


(b) 419 cells, $p_c^s = 0.45$

Figure 3.12: *Site percolation for (a) 426 cells, no constraints, (b) 419 cells, limit* $= 0.01$



(a) 416 cells, $p_c^s = 0.50$

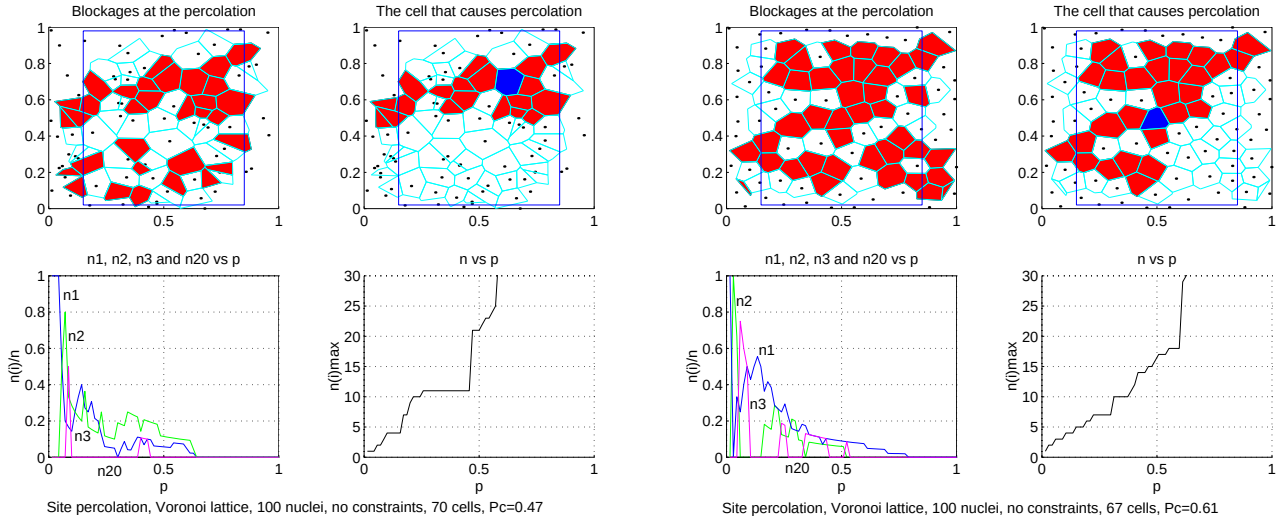


(b) 413 cells, $p_c^s = 0.53$

Figure 3.13: *Site percolation for (a) 416 cells, limit* $= 0.02$*, (b) 413 cells, limit* $= 0.03$

3.2 Algorithm using nearest neighbours

The following are the results of 2 simulations done by using Algorithm 2 in order to try a different algorithm and compare with the results shown in the last section. It turns out that they produce very similar results, therefore only 2 run results are included here.



(a) $p_c^s = 0.47$ *(b)* $p_c^s = 0.61$

Figure 3.14: *Site percolation for (a) 70 cells, no constraints, (b) 67 cells, no constraints*

3.3 Discussion

Each figure shows some variation of p_c^s with higher variation for smaller networks. This is due to the fact that the networks are random in nature. The value of p_c^s for discussion later on will be the arithmetic mean value from several runs.

The shapes of the percolating clusters indicate side branches which do not contribute directly to the percolating path. These side branches occur because the cells are blocked randomly, and depend on the structure of the networks. Good examples of these branches can be seen in Figure 3.1*(a)*, 3.2, 3.3, 3.6, 3.7, 3.9 - 3.13, and 3.14*(b)*. The higher the tendency of a network in forming these side branches (or redundancies), the higher the value of p_c will be.

The plot of percentage of clusters of a given size against p shows that the maximum percentage for clusters with a larger size occurs at higher value of p. If we draw the plot for every size, we should get a family of curves distributed symmetrically on both side of $p = p_c$.

The plot of maximum cluster sizes against p shows a characteristic jump at the value $p = p_c$ which signifies the idea of *phase transition* where there is a phase reversal between being mostly occupied and being mostly vacant.

The $\bar{p_c^s}$, from 27 simulations on networks of various sizes, was found to be 0.507. The two algorithms used gave the same mean value of p_c^s ($\bar{p_c^s}$), so the next-nearest neighbours algorithm (Algorithm 1) is prefered here because it is easier to program and is faster to run. However, with degeneraed lattices, where there are many sites with coordination number ≥ 4, one would have to use Algorithm 2 in order to obtain a correct result.

Chapter 4

Bond percolation results of 2-D Voronoi networks

Our Voronoi networks are created from nuclei randomly placed within a bounded region of unit area. Along the rim of such area is the region where cells produced are mis-shaped due to the fact that they cannot grow past the boundary. We discard all such cells, thus leaving us a Voronoi networks with an area somewhat less than 1. When studying bond percolation, we draw 2 imaginary lines $x = constant$ each along the left and the right edge of the Voronoi region. We say that percolation occurs when there is a single cluster intersecting both lines. The two lines, together with the lines $y = \epsilon$ and $y = 1 - \epsilon$, where $\epsilon \in \mathbb{R}$ and $\epsilon \ll 1$, form a control area. Let A_c be our control area, so that $A_c < 1$ and the size of our Voronoi networks depends upon the number of cells or bonds in it. As the size of the networks increase the span of the percolating cluster will tend to infinity and the p_c obtained from simulation will get closer to the true value of p_c for an infinite cluster.

Two different algorithms were used to obtain the bond critical probability. The first algorithm (Algorithm 3) takes in to account every bond that forms the networks, while the second algorithm (Algorithm 4) considers only those not lying completely within our control area. In other words, two different definitions of p_c^b were used in the simulations, namely

Definition 4.1 *For Algorithm 3 (edge approximation),*

$$p_c^b = \frac{number\ of\ blocked\ bonds}{total\ number\ of\ bonds}$$

where p_c^b is the bond critical probability.

and

Definition 4.2 *For Algorithm 4 (control area),*

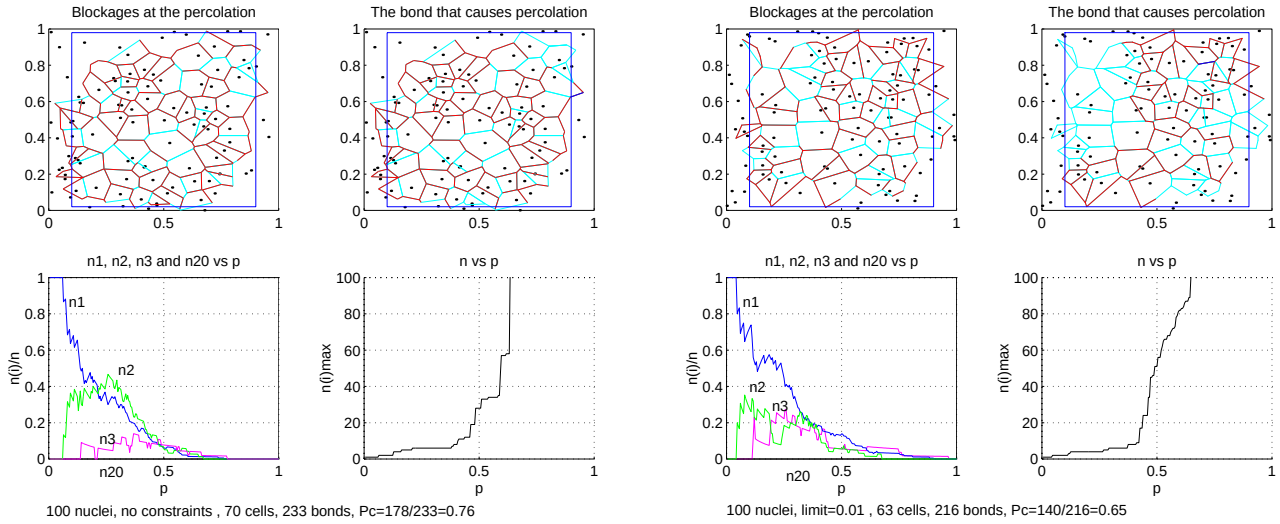
$$p_c^b = \frac{number\ of\ the\ blocked\ bonds\ within\ the\ control\ area}{total\ number\ of\ bonds\ within\ the\ control\ area}$$

where p_c^b is the bond critical probability.

For some of the pictures, $n = total\ number\ of\ blocked\ bonds$, $n(i) = number\ of\ bonds\ in\ cluster\ i$, $n(i)_{max} = number\ of\ bonds\ in\ the\ largest\ cluster$.

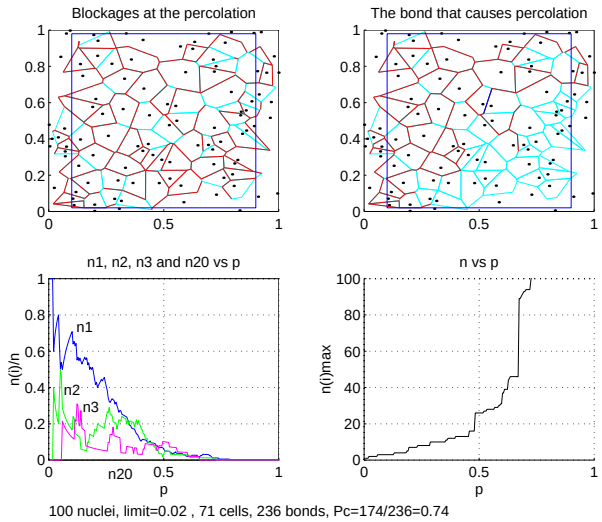
4.1 Algorithm using edge approximation

The following simulations (Figures 4.1 - 4.25) were done by using Algorithm 3. Some were done using a program written in C; the rest were done using a program in MATLAB (see Appendix B for details).

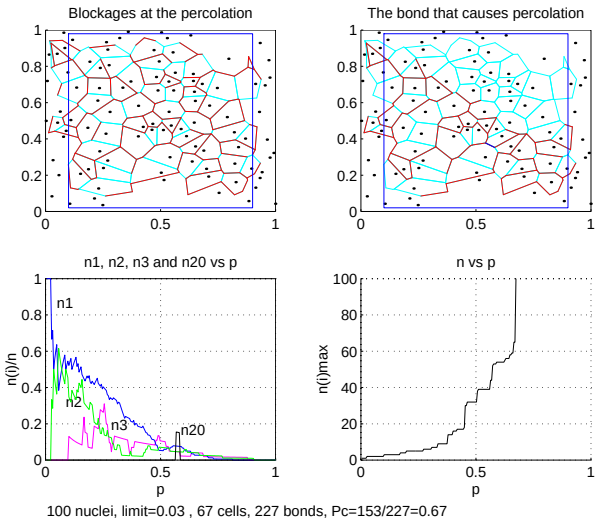


(a) 233 bonds, $P_c^b = 0.76$ *(b) 216 bonds, $P_c^b = 0.65$*

Figure 4.1: *Bond percolation for (a) 233 bonds, no constraints, (b) 216 bonds, limit* $= 0.01$

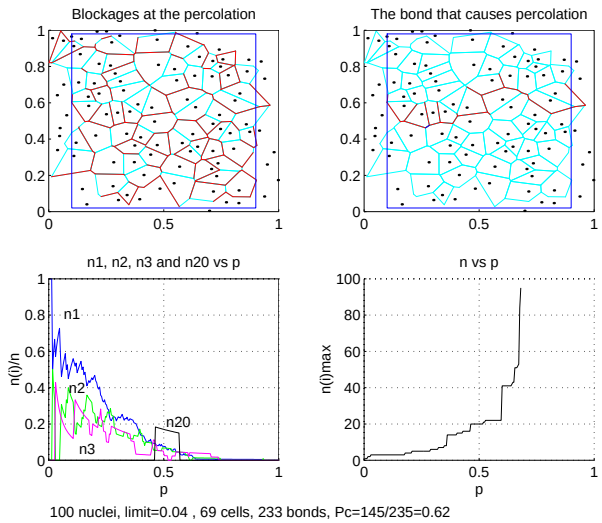


(a) 236 bonds, $P_c^b = 0.76$

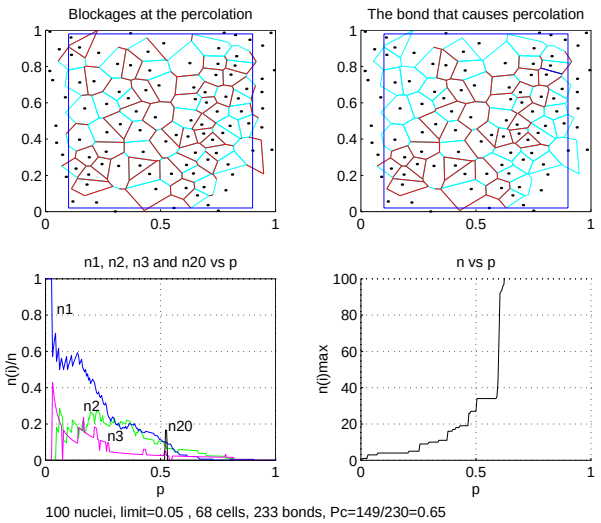


(b) 227 bonds, $P_c^b = 0.65$

Figure 4.2: *Bond percolation for (a) 236 bonds, limit* $= 0.02$*, (b) 227 bonds, limit* $= 0.03$

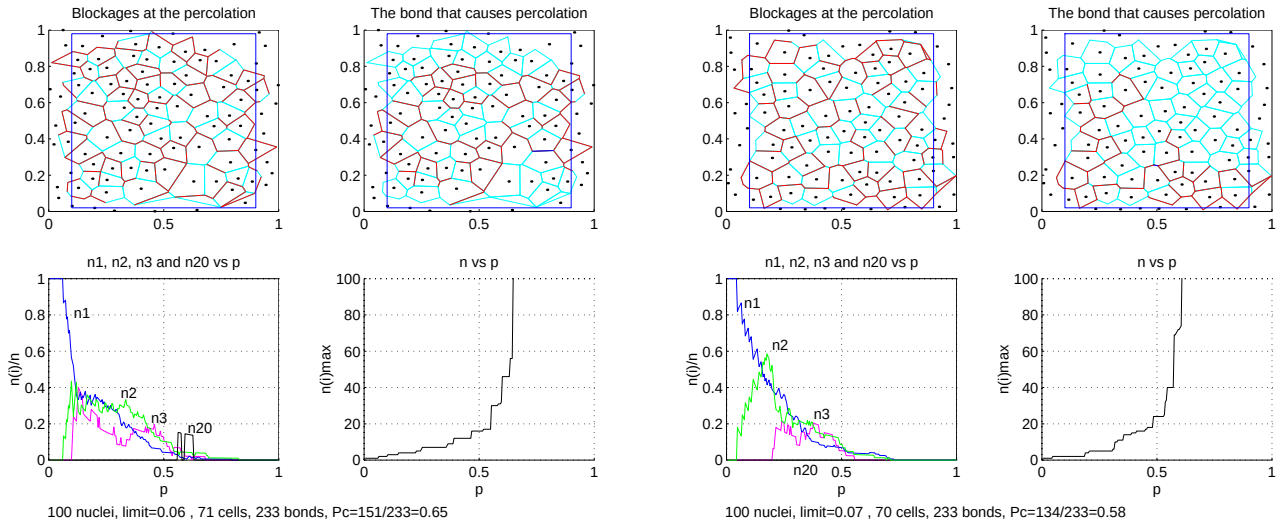


(a) 235 bonds, $P_c^b = 0.62$



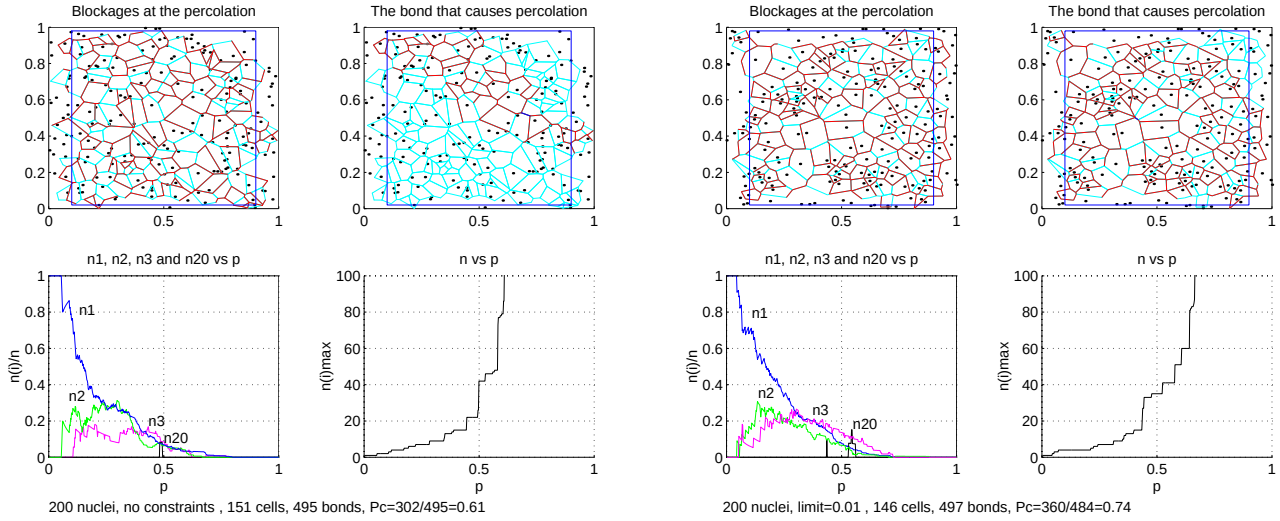
(b) 230 bonds, $P_c^b = 0.65$

Figure 4.3: *Bond percolation for (a) 233 bonds, limit* $= 0.04$*, (b) 233 bonds, limit* $= 0.05$



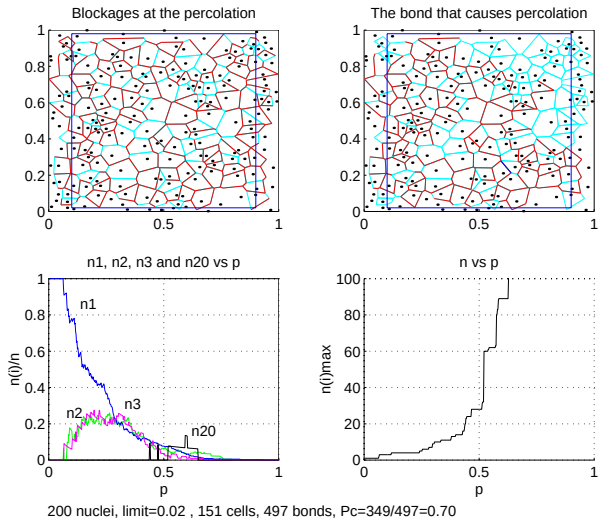
(a) 233 bonds, $P_c^b = 0.65$ *(b) 233 bonds,* $P_c^b = 0.58$

Figure 4.4: *Bond percolation for (a) 233 bonds, limit* $= 0.06$*, (b) 233 bonds, limit* $= 0.07$

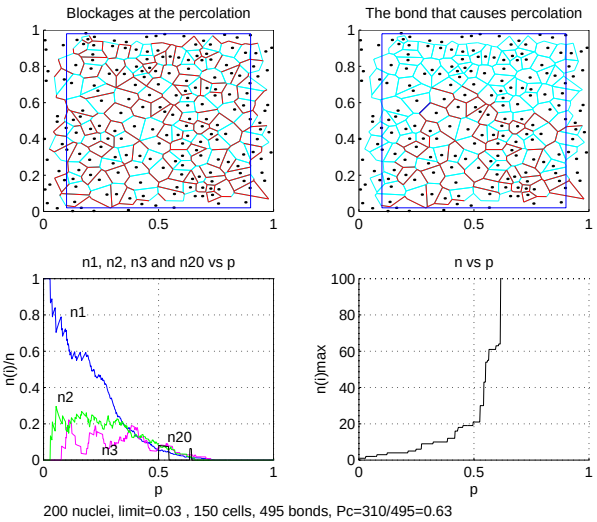


(a) 495 bonds, $P_c^b = 0.61$ *(b) 484 bonds,* $P_c^b = 0.74$

Figure 4.5: *Bond percolation for (a) 495 bonds, no constraints, (b) 497 bonds, limit* $= 0.01$

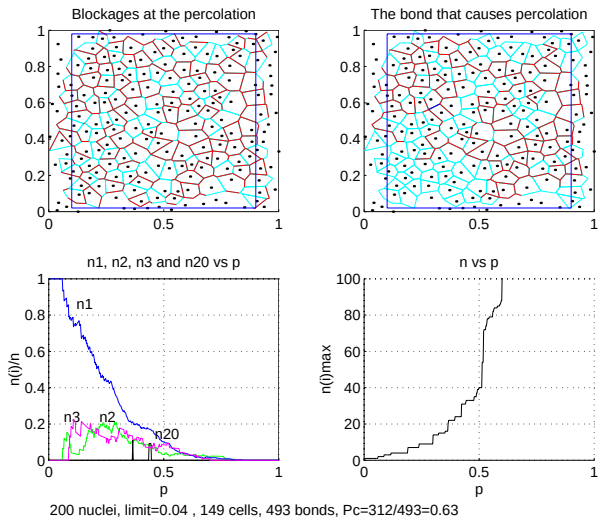


(a) 497 bonds, $P_c^b = 0.70$

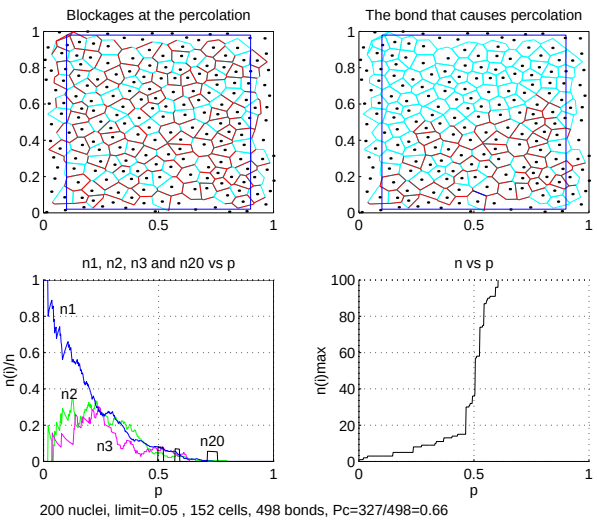


(b) 495 bonds, $P_c^b = 0.63$

Figure 4.6: *Bond percolation for (a) 497 bonds, limit* $= 0.02$*, (b) 495 bonds, limit* $= 0.03$

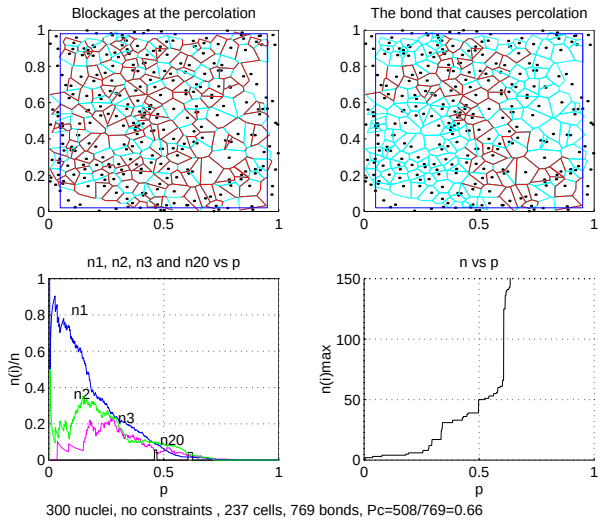


(a) 493 bonds, $P_c^b = 0.63$

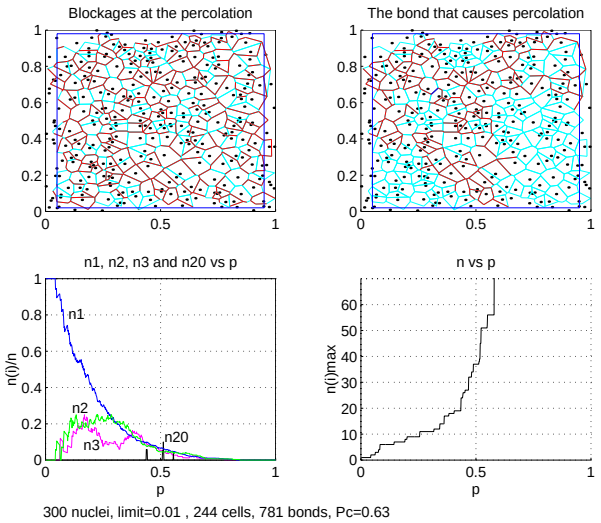


(b) 498 bonds, $P_c^b = 0.66$

Figure 4.7: *Bond percolation for (a) 493 bonds, limit* $= 0.04$*, (b) 498 bonds, limit* $= 0.05$

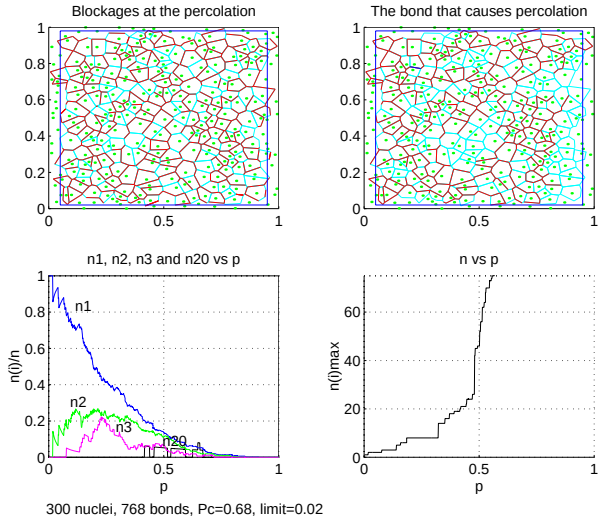


(a) 769 bonds, $P_c^b = 0.66$

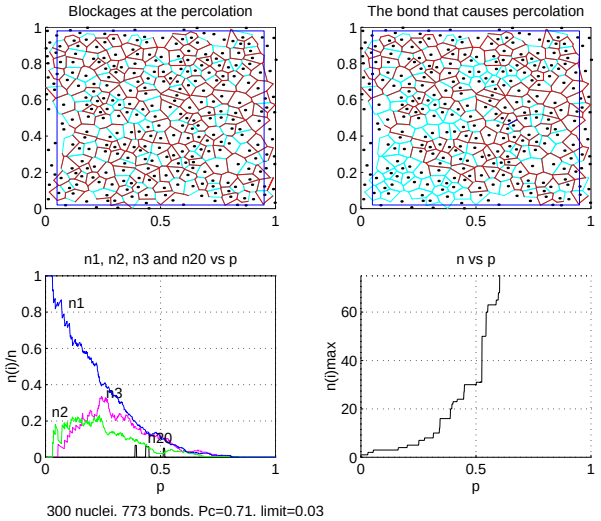


(b) 781 bonds, $P_c^b = 0.63$

Figure 4.8: *Bond percolation for (a) 769 bonds, no constraints, (b) 781 bonds, limit* $= 0.01$

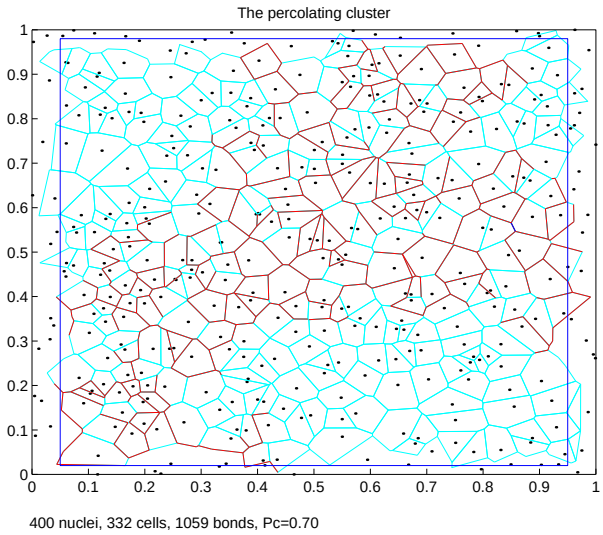


(a) 768 bonds, $P_c^b = 0.68$

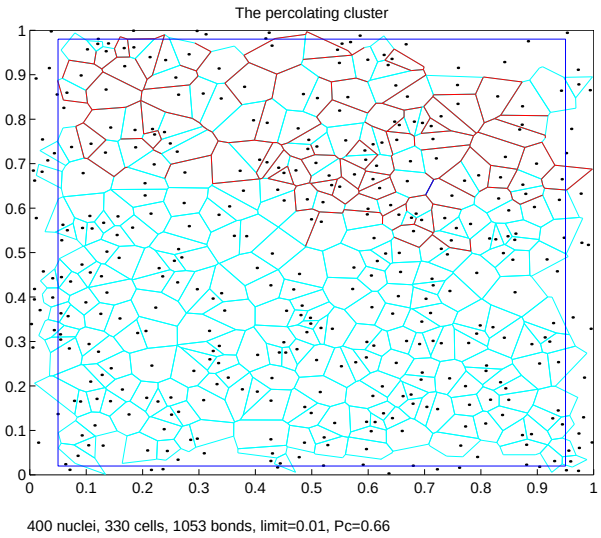


(b) 773 bonds, $P_c^b = 0.71$

Figure 4.9: *Bond percolation for (a) 768 bonds, limit* $= 0.02$*, (b) 773 bonds, limit* $= 0.03$

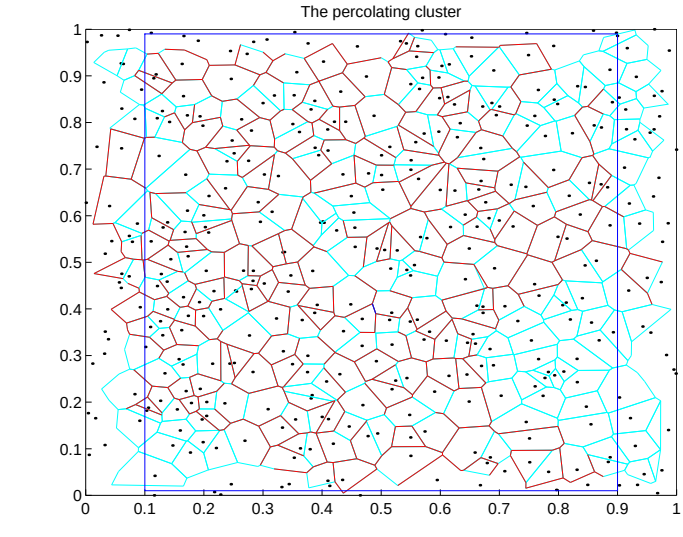


(a) 1059 bonds, $P_c^b = 0.70$



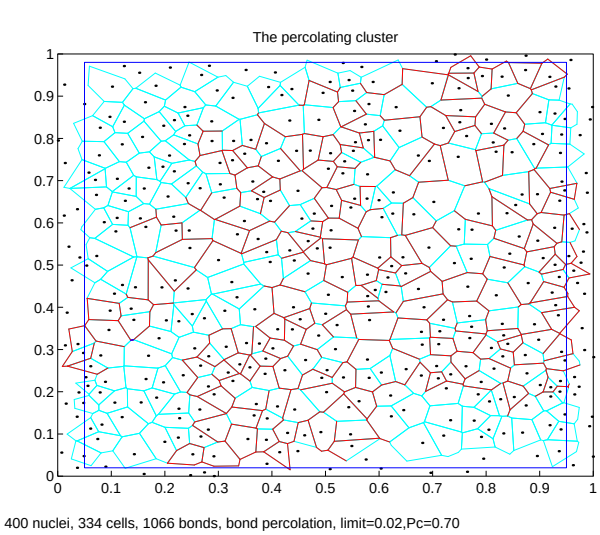
(b) 1053 bonds, $P_c^b = 0.66$

Figure 4.10: *Bond percolation for (a) 1059 bonds, no constraints, (b) 1053 bonds, limit* $= 0.01$

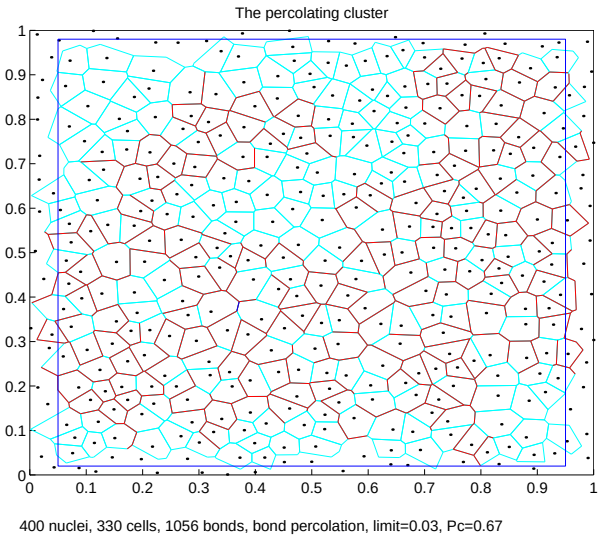


1059 bonds, $P_c^b = 0.67$

Figure 4.11: *Bond percolation for 1059 bonds, no constraints*

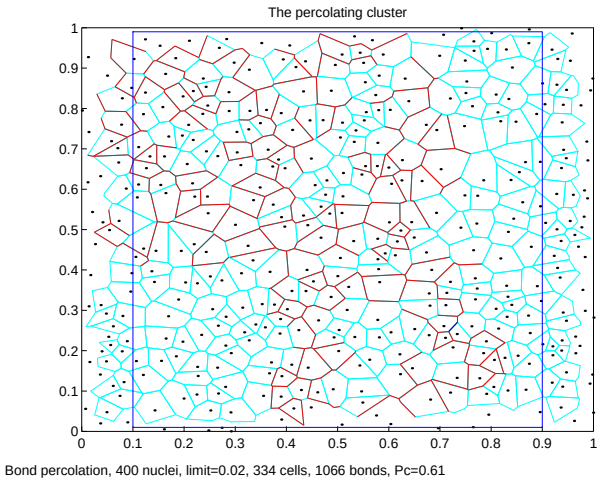


(a) 1066 bonds, $P_c^b = 0.70$

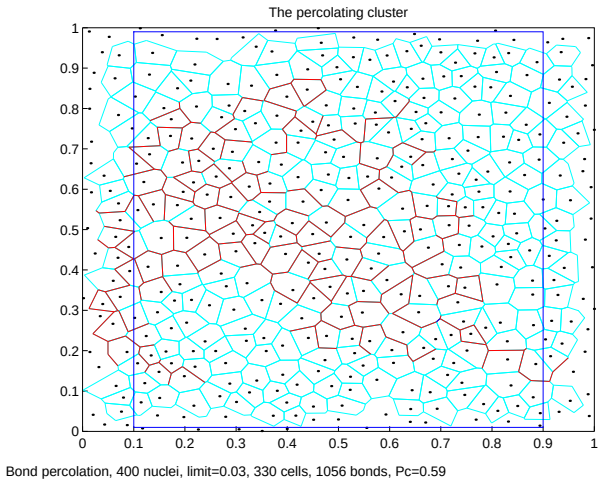


(b) 1056 bonds, $P_c^b = 0.67$

Figure 4.12: *Bond percolation for (a) 1066 bonds, limit* $= 0.02$*, (b) 1056 bonds, limit* $= 0.03$

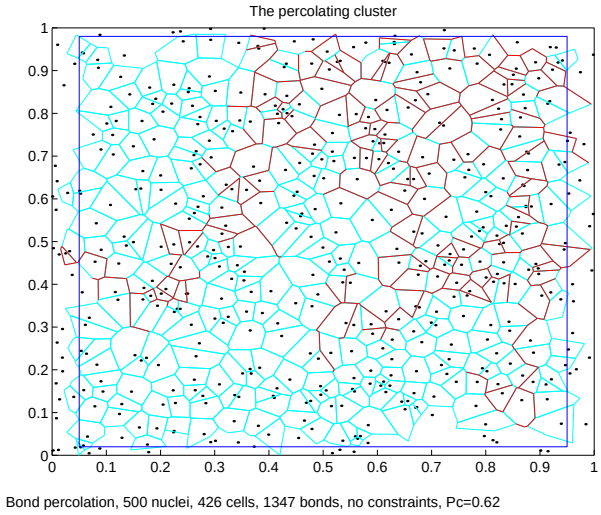


(a) 1066 bonds, $P_c^b = 0.61$

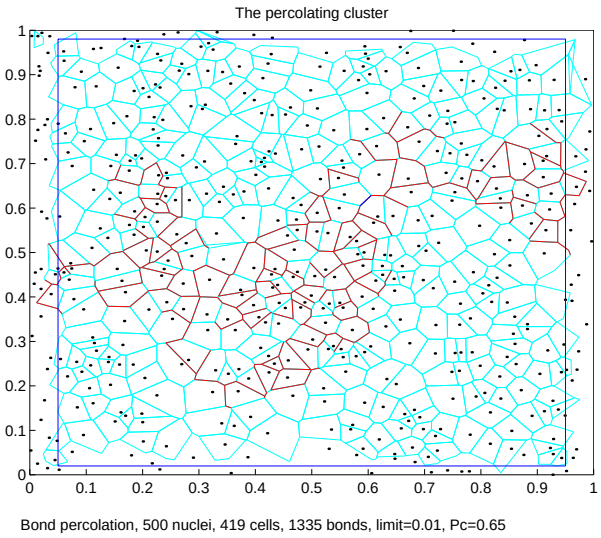


(b) 1056 bonds, $P_c^b = 0.59$

Figure 4.13: *Bond percolation for (a) 1066 bonds, limit* $= 0.02$*, (b) 1056 bonds, limit* $= 0.03$

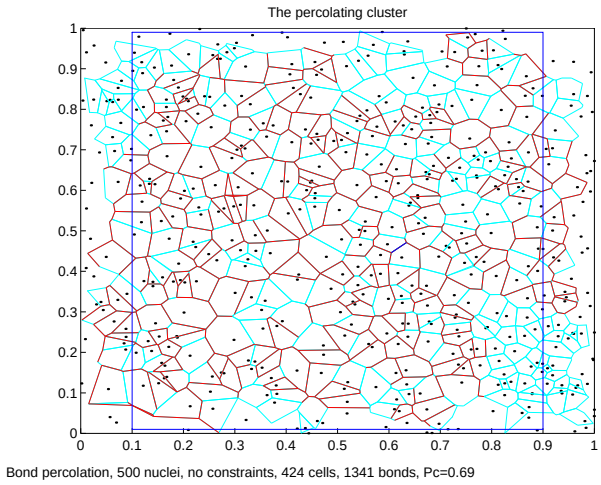


(a) 1347 bonds, $P_c^b = 0.62$

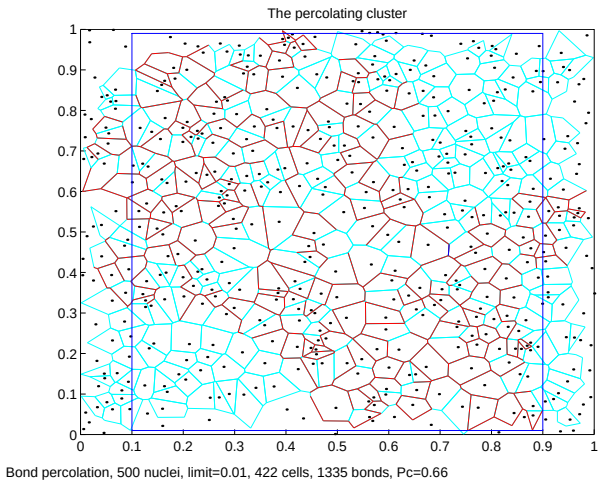


(b) 1335 bonds, $P_c^b = 0.65$

Figure 4.14: *Bond percolation for (a) 1347 bonds, no constraints, (b) 1335 bonds, limit* $= 0.01$

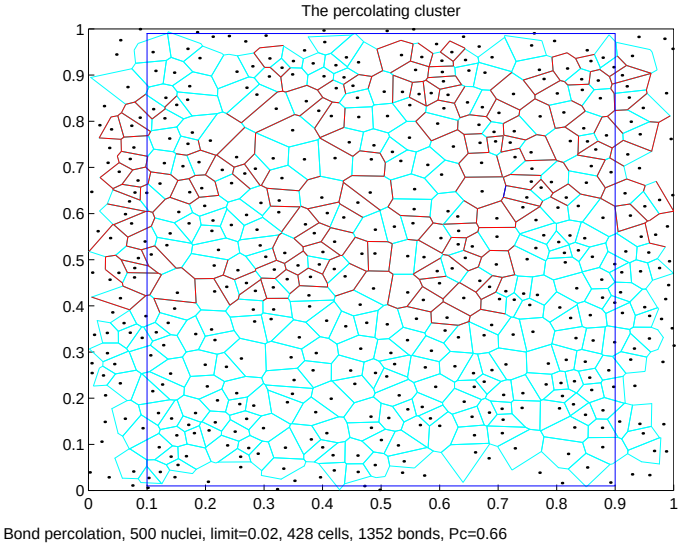


(a) 1341 bonds, $P_c^b = 0.69$



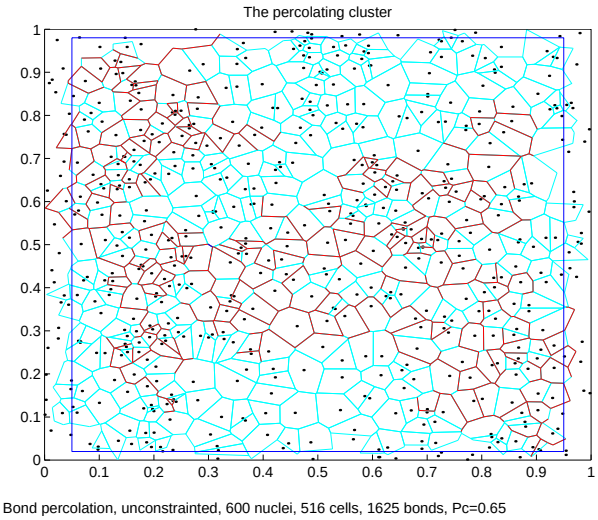
(b) 1335 bonds, $P_c^b = 0.66$

Figure 4.15: *Bond percolation for (a) 1341 bonds, no constraints, (b) 1335 bonds, limit* $= 0.01$

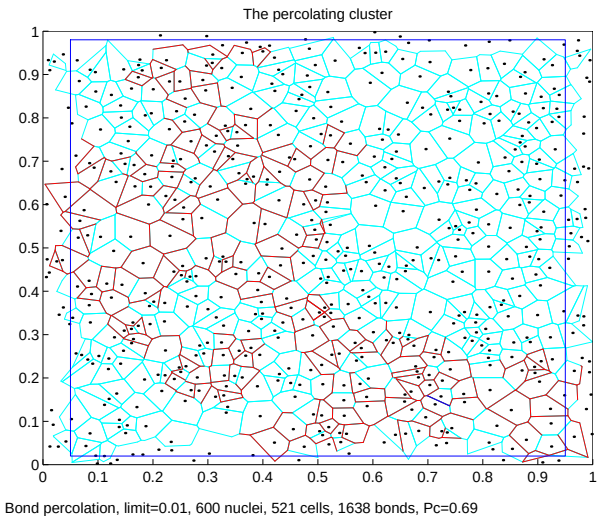


1352 bonds, $P_c^b = 0.66$

Figure 4.16: *Bond percolation for 1352 bonds, limit* $= 0.02$

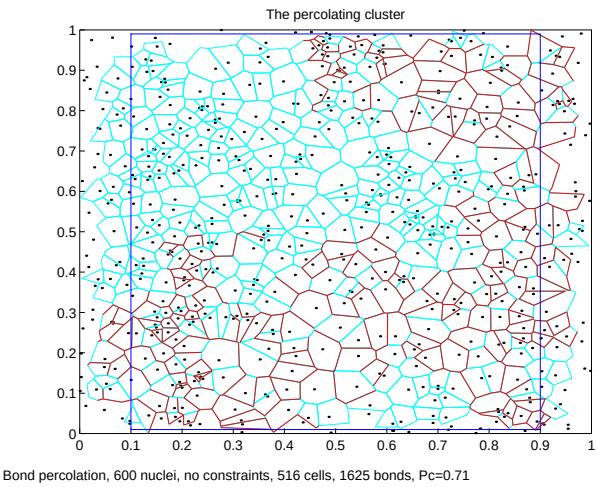


(a) 1625 bonds, $P_c^b = 0.65$

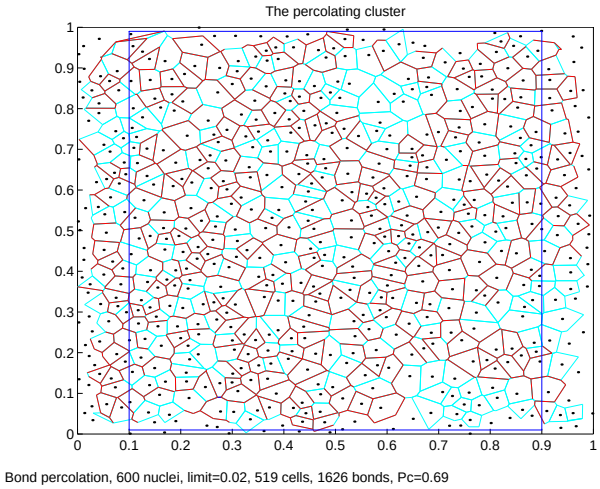


(b) 1638 bonds, $P_c^b = 0.69$

Figure 4.17: *Bond percolation for (a) 1625 bonds, no constraints, (b) 1638 bonds, limit* $= 0.01$

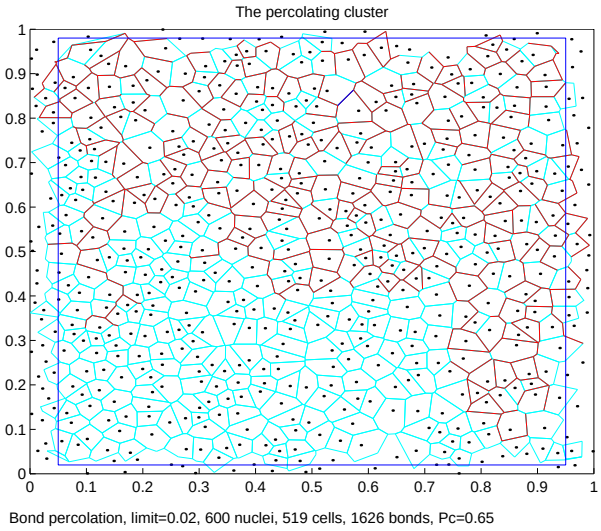


(a) 1625 bonds, $P_c^b = 0.71$

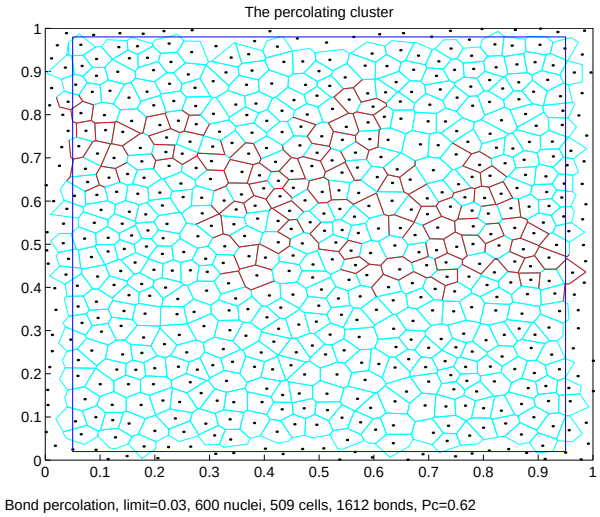


(b) 1626 bonds, $P_c^b = 0.69$

Figure 4.18: *Bond percolation for (a) 1625 bonds, no constraints, (b) 1626 bonds, limit* $= 0.02$

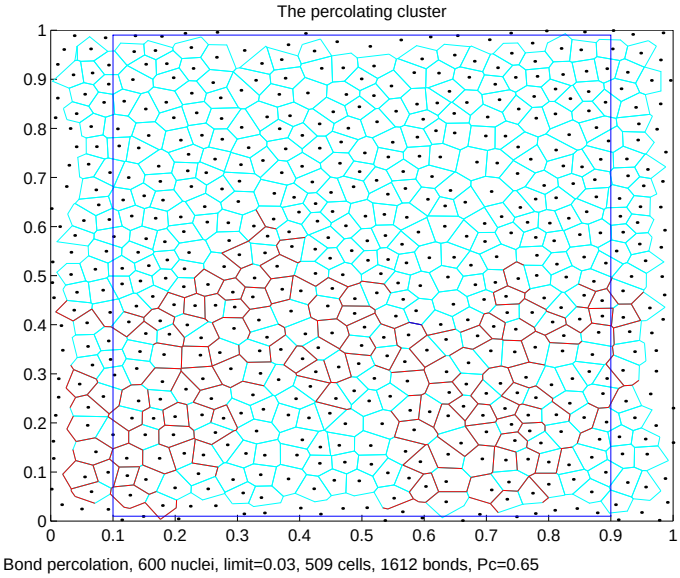


(a) 1626 bonds, $P_c^b = 0.65$



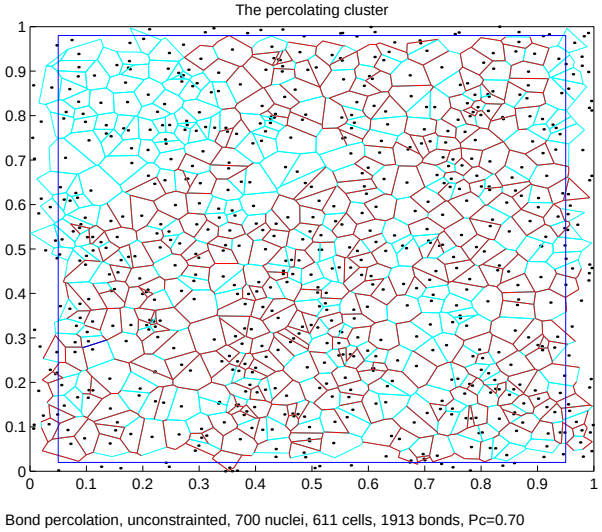
(b) 1612 bonds, $P_c^b = 0.62$

Figure 4.19: *Bond percolation for (a) 1626 bonds, limit* $= 0.02$*, (b) 1612 bonds, limit* $= 0.03$

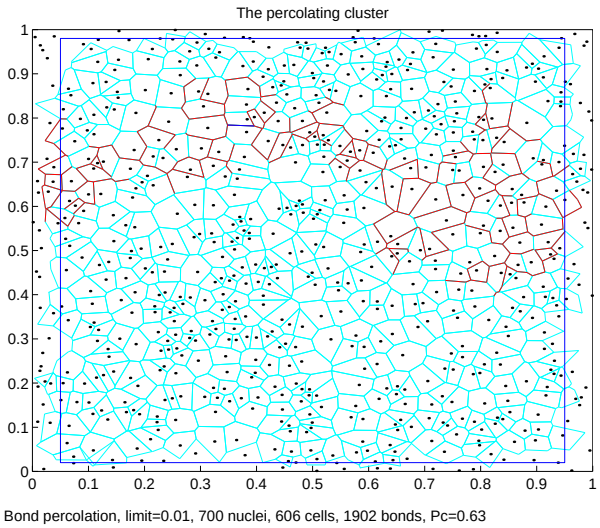


1612 bonds, $P_c^b = 0.65$

Figure 4.20: *Bond percolation for 1612 bonds, limit* $= 0.03$

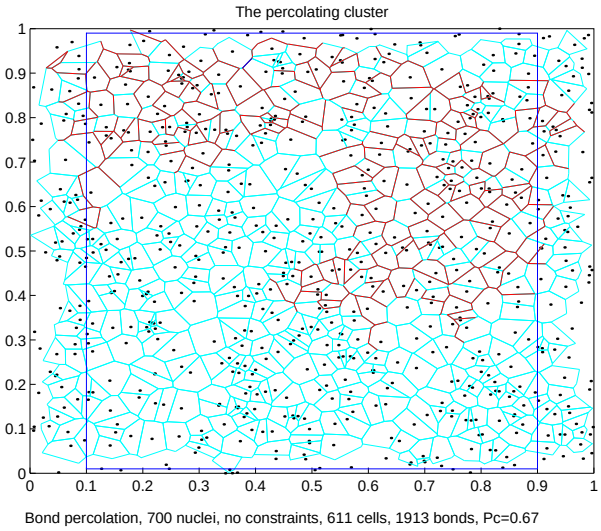


(a) 1913 bonds, $P_c^b = 0.70$

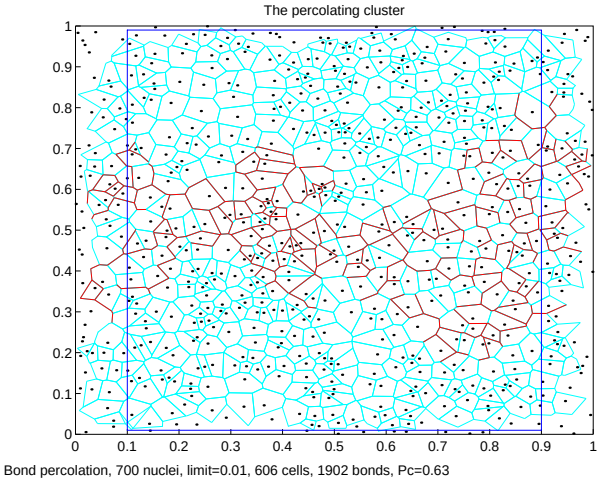


(b) 1902 bonds, $P_c^b = 0.63$

Figure 4.21: *Bond percolation for (a) 1913 bonds, no constraints, (b) 1902 bonds, limit* $= 0.01$

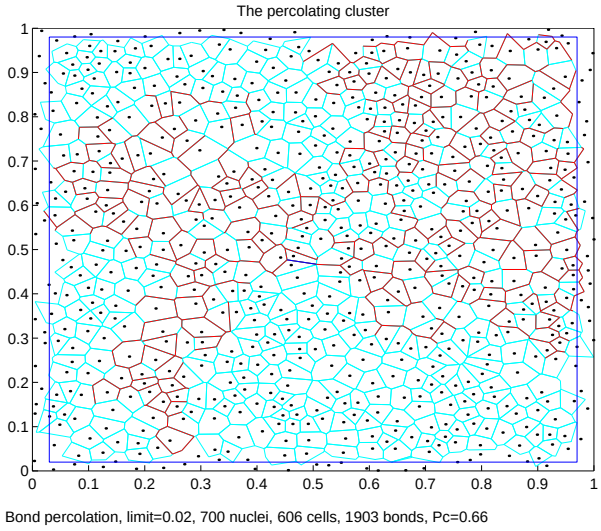


(a) 1913 bonds, $P_c^b = 0.67$

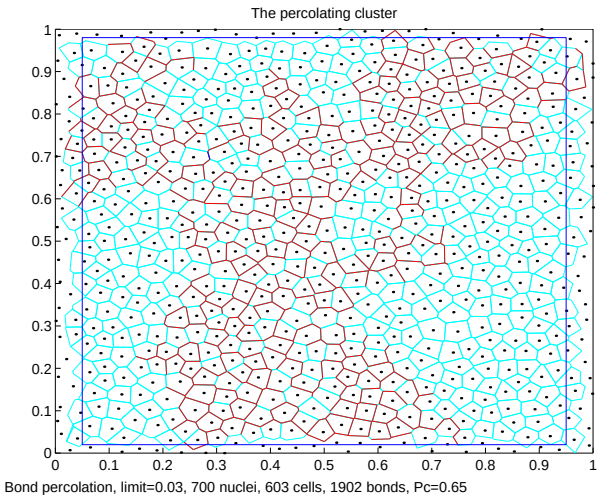


(b) 1902 bonds, $P_c^b = 0.63$

Figure 4.22: *Bond percolation for (a) 1913 bonds, no constraints, (b) 1902 bonds, limit* $= 0.01$

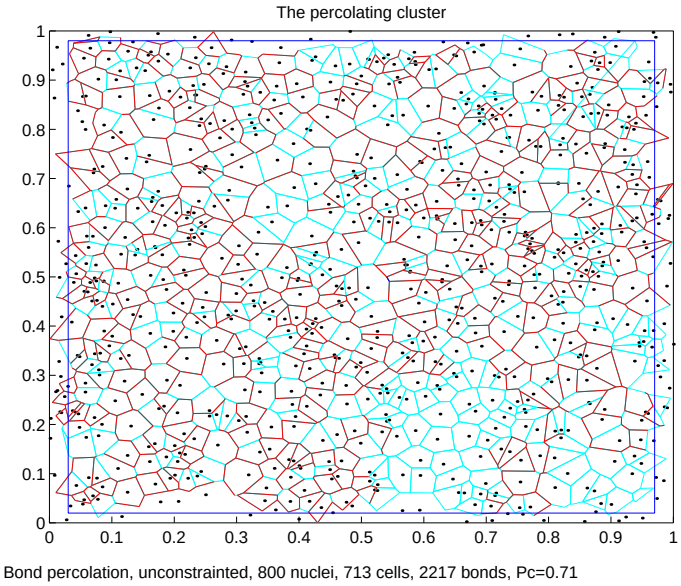


(a) 1913 bonds, $P_c^b = 0.66$



(b) 1902 bonds, $P_c^b = 0.65$

Figure 4.23: *Bond percolation for (a) 1913 bonds, limit* $= 0.02$*, (b) 1902 bonds, limit* $= 0.03$



2217 bonds, $P_c^b = 0.71$

Figure 4.24: *Bond percolation for 2217 bonds, no constraints*

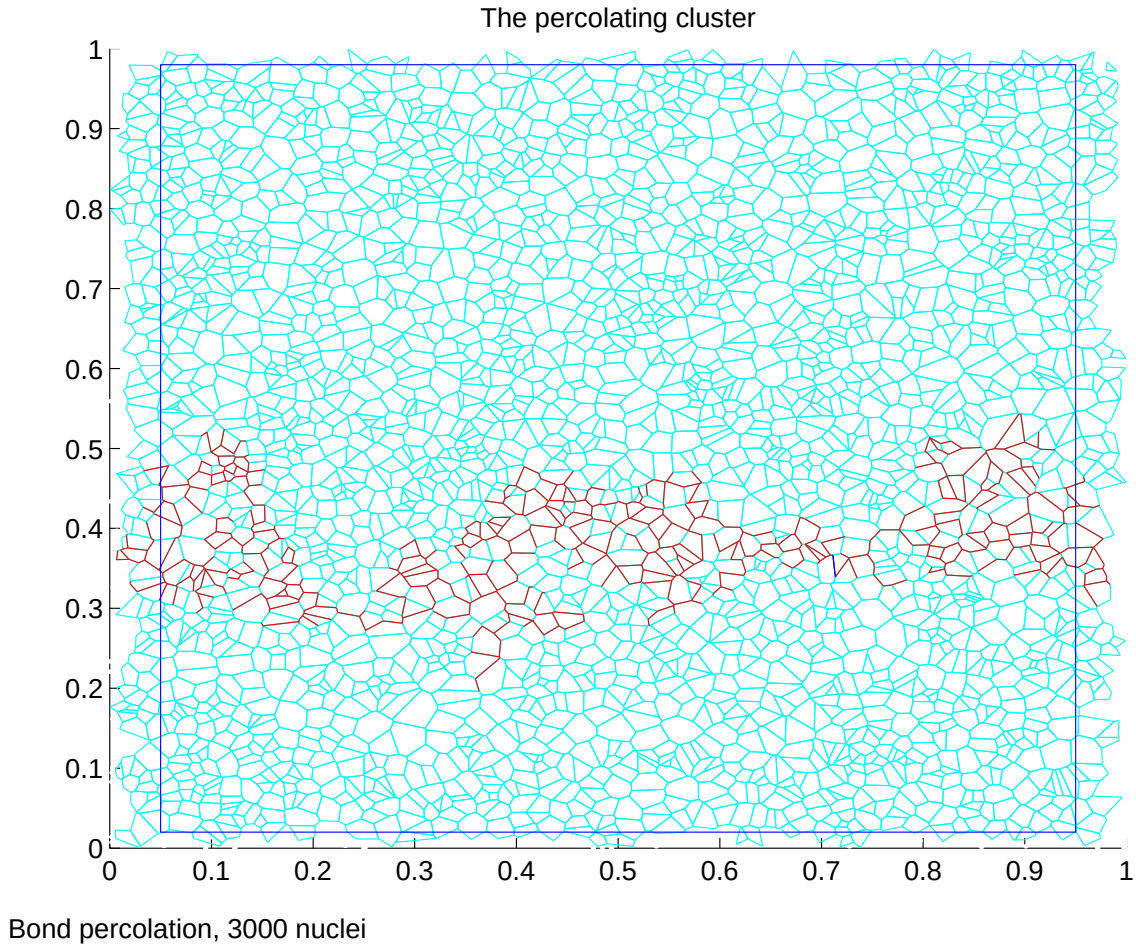
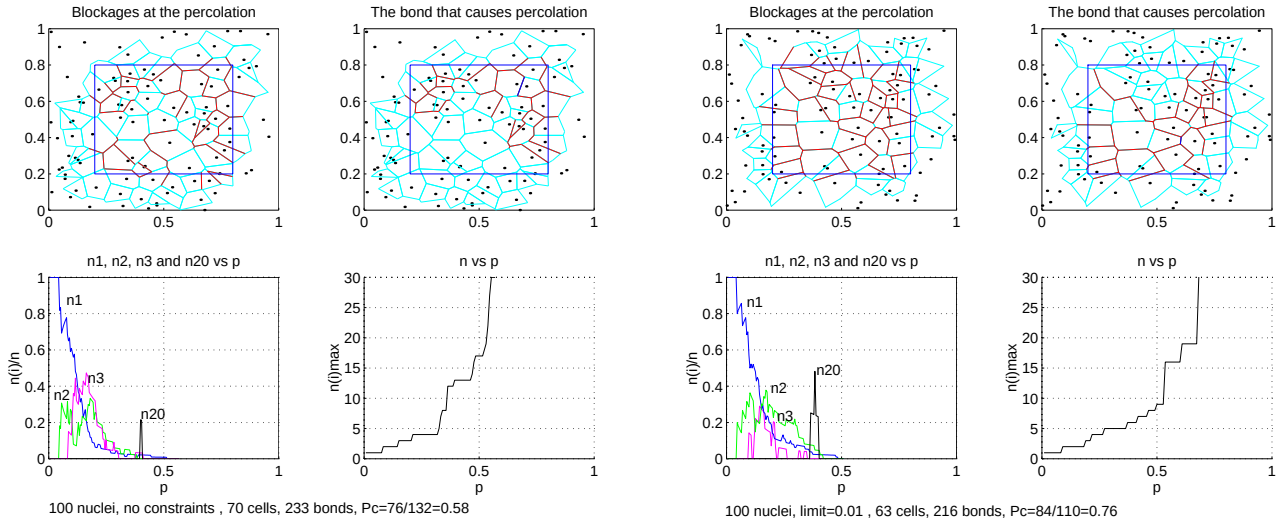


Figure 4.25: *Bond percolation for 8609 bonds, no constraints*

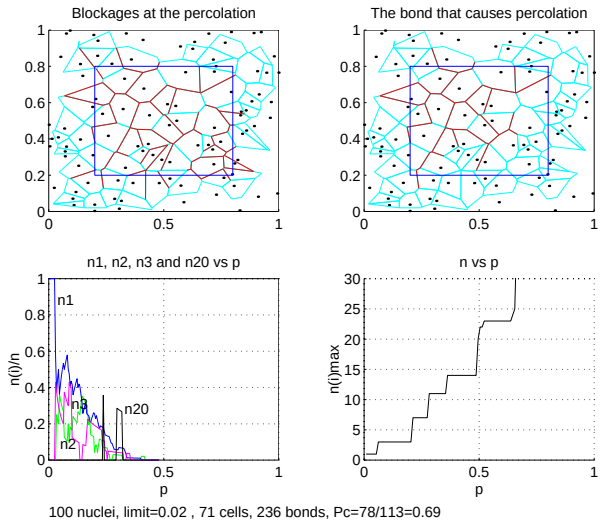
4.2 Algorithm using a control area

The following results (Figures 4.26 - 4.34) were obtained using Algorithm 4 which takes into account only those bonds which are within a specified area.

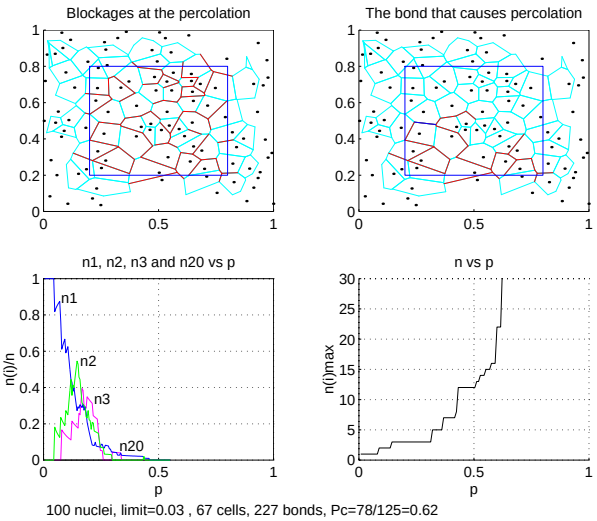


(a) 132/233 *bonds,* $P_c^b = \frac{76}{132} = 0.58$ *(b)* 110/216 *bonds,* $P_c^b = \frac{84}{110} = 0.76$

Figure 4.26: *Bond percolation for (a) 132 bonds, no constraints, (b) 110 bonds, limit* $= 0.01$

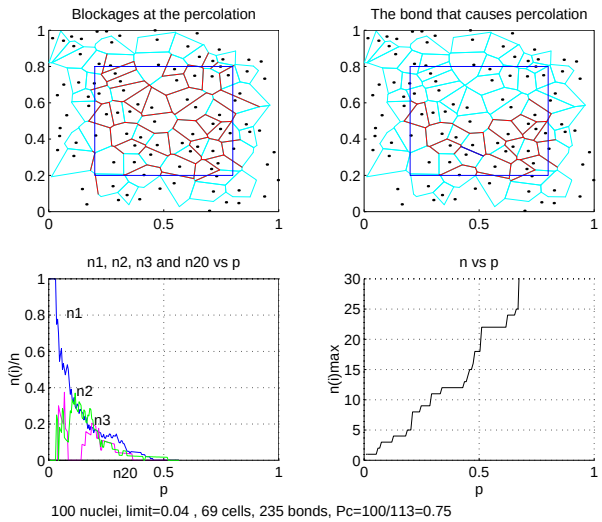


(a) 113/236 *bonds,* $P_c^b = \frac{78}{113} = 0.69$

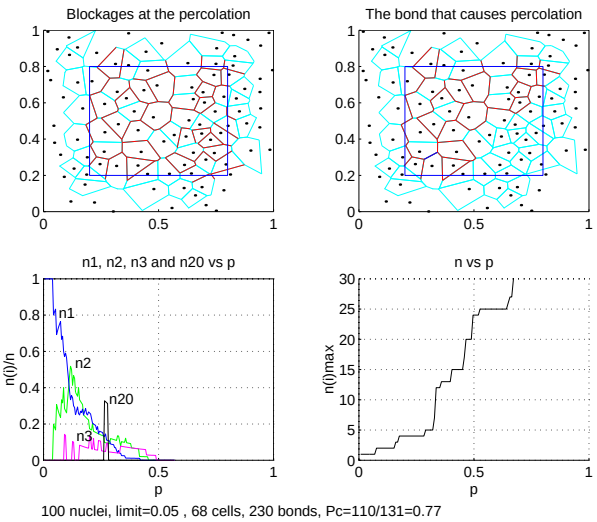


(b) 125/227 *bonds,* $P_c^b = \frac{78}{125} = 0.62$

Figure 4.27: *Bond percolation for (a) 113 bonds, limit* $= 0.02$*, (b) 125 bonds, limit* $= 0.03$

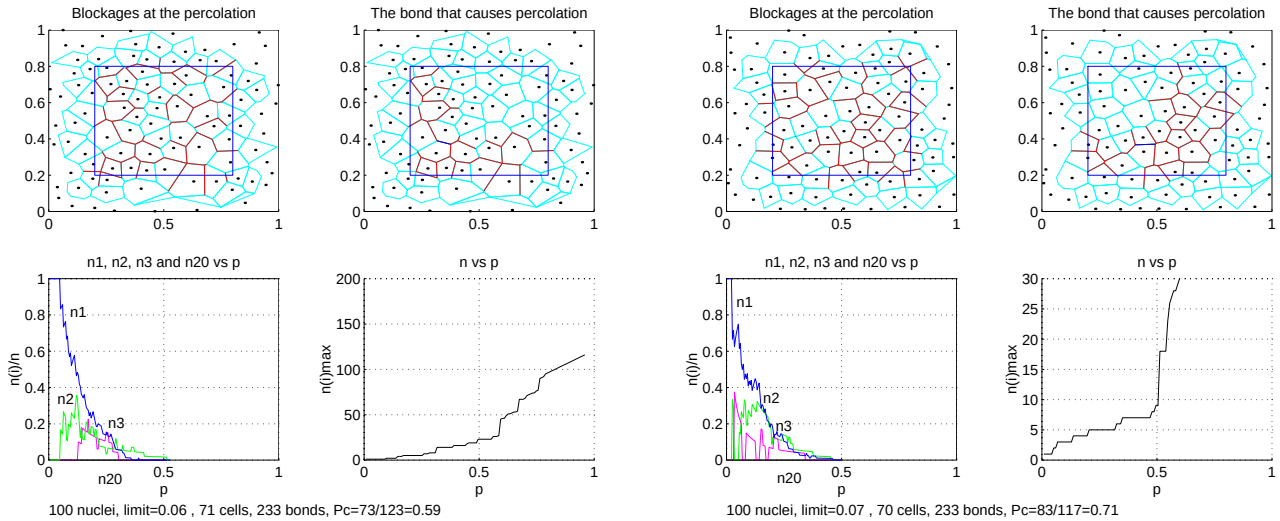


(a) 113/235 *bonds,* $P_c^b = \frac{100}{113} = 0.75$



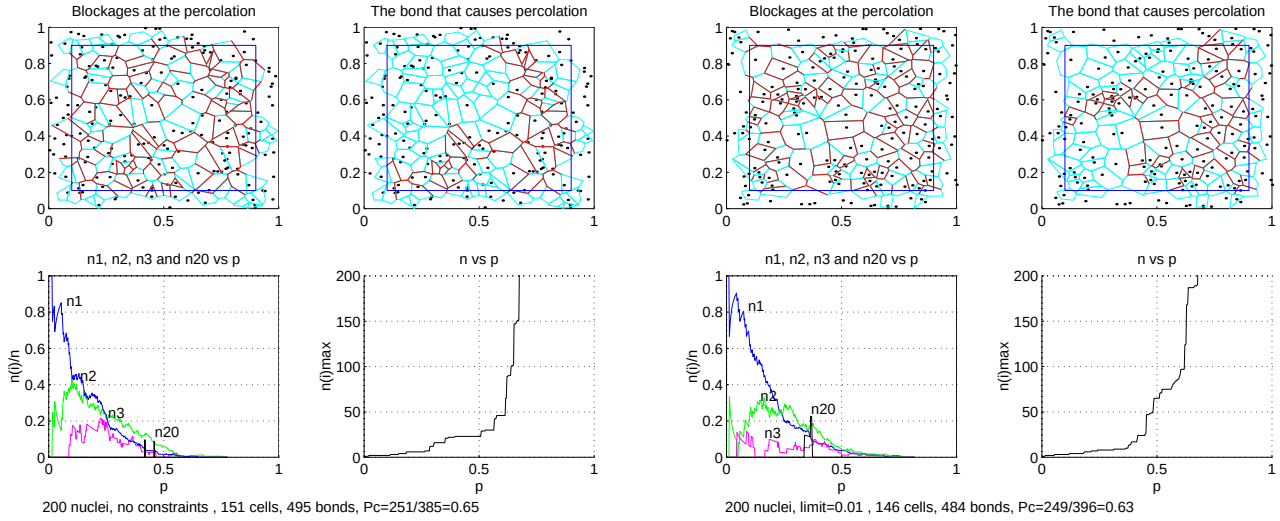
(b) 131/230 *bonds,* $P_c^b = \frac{110}{131} = 0.77$

Figure 4.28: *Bond percolation for (a) 113 bonds, limit* $= 0.04$*, (b) 131 bonds, limit* $= 0.05$



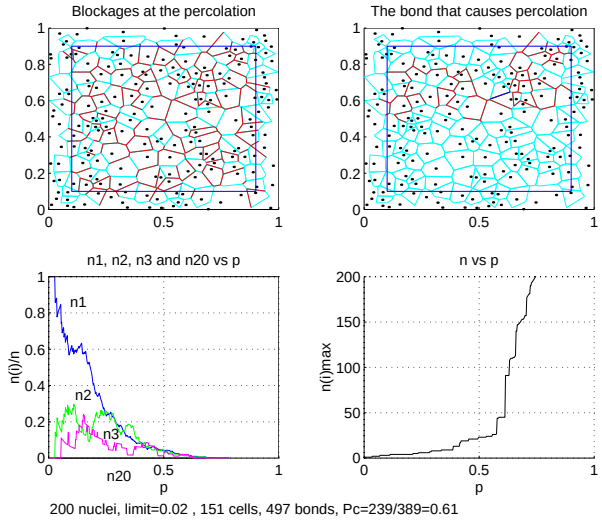
(a) 123/233 *bonds,* $P_c^b = \frac{73}{123} = 0.59$ *(b)* 117/233 *bonds,* $P_c^b = \frac{83}{117} = 0.71$

Figure 4.29: *Bond percolation for (a) 123 bonds, limit* $= 0.06$*, (b) 117 bonds, limit* $= 0.07$

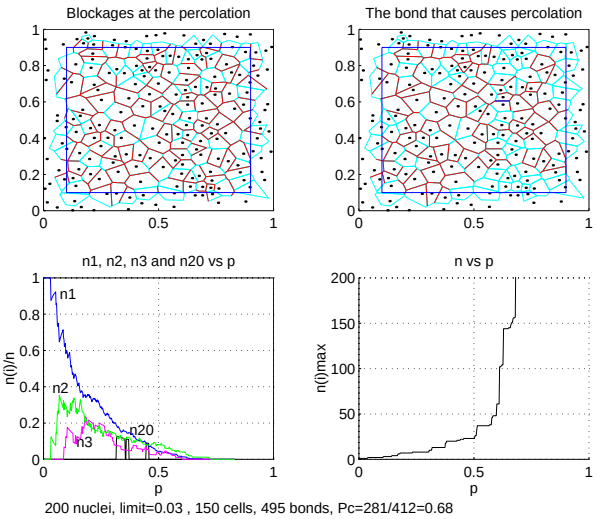


(a) 385/495 *bonds,* $P_c^b = \frac{251}{385} = 0.65$ *(b)* 396/484 *bonds,* $P_c^b = \frac{249}{396} = 0.63$

Figure 4.30: *Bond percolation for (a) 385 bonds, no constraints, (b) 396 bonds, limit* $= 0.01$

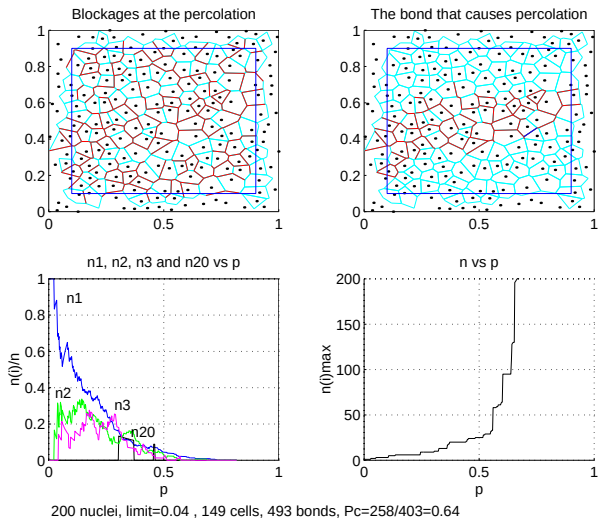


(a) 389/497 *bonds,* $P_c^b = \frac{239}{389} = 0.61$

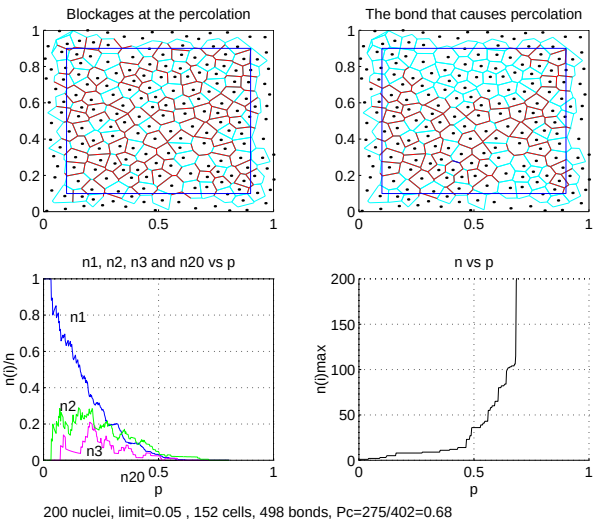


(b) 412/495 *bonds,* $P_c^b = \frac{281}{412} = 0.68$

Figure 4.31: *Bond percolation for (a) 389 bonds, limit* $= 0.02$*, (b) 412 bonds, limit* $= 0.03$

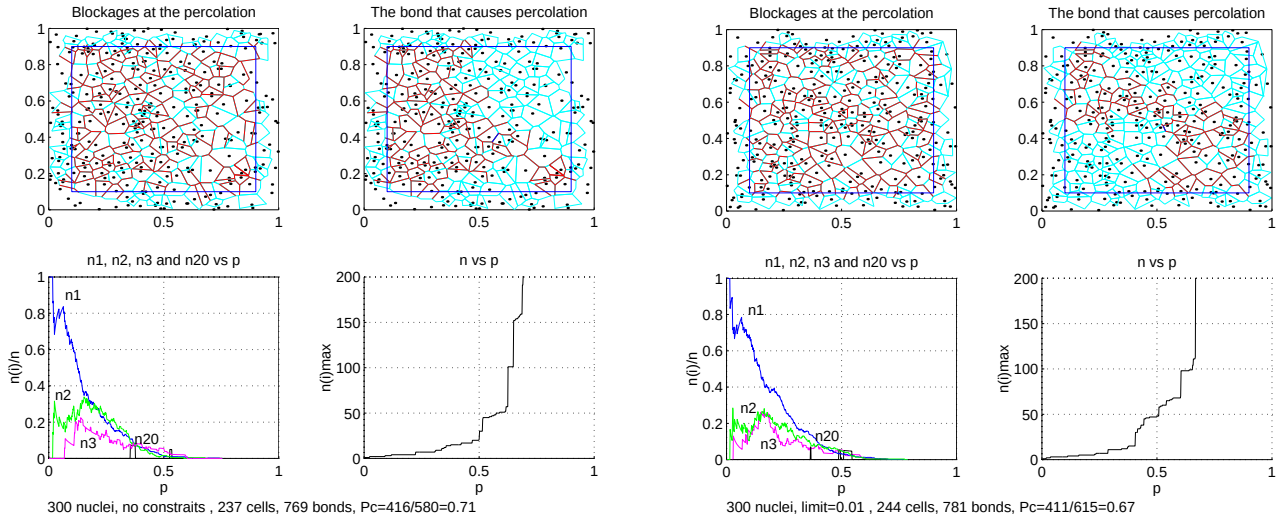


(a) 403/493 *bonds,* $P_c^b = \frac{258}{403} = 0.64$



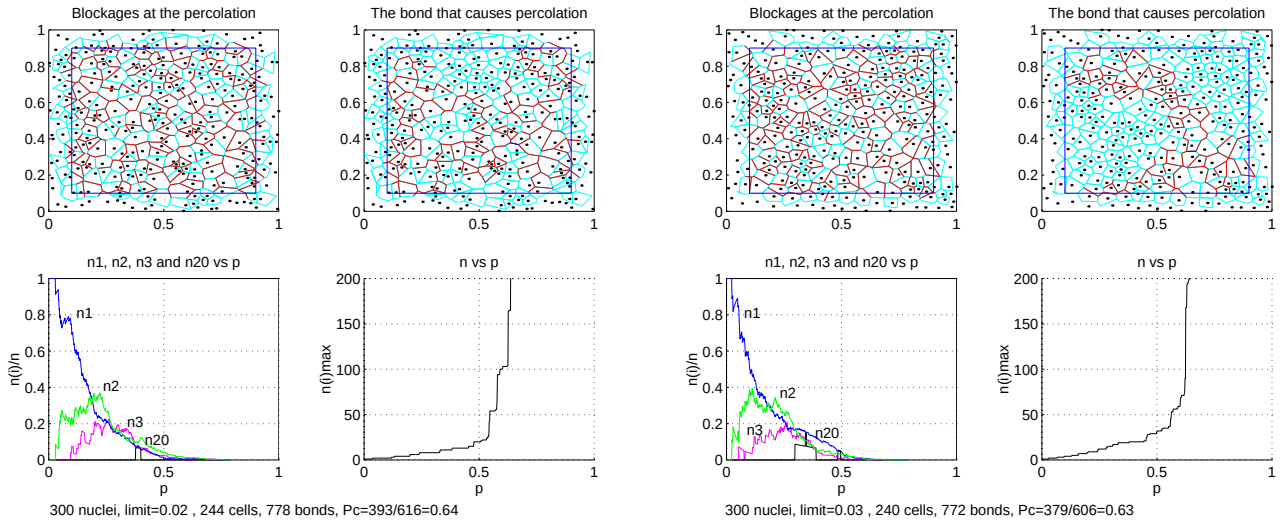
(b) 402/498 *bonds,* $P_c^b = \frac{275}{402} = 0.68$

Figure 4.32: *Bond percolation for (a) 403 bonds, limit* $= 0.04$*, (b) 402 bonds, limit* $= 0.05$



(a) 580/769 *bonds,* $P_c^b = \frac{416}{580} = 0.71$ *(b)* 615/781 *bonds,* $P_c^b = \frac{411}{615} = 0.67$

Figure 4.33: *Bond percolation for (a) 580 bonds, no constraints, (b) 615 bonds, limit* $= 0.01$



(a) 616/778 *bonds,* $P_c^b = \frac{393}{616} = 0.64$ *(b)* 606/772 *bonds,* $P_c^b = \frac{379}{606} = 0.63$

Figure 4.34: *Bond percolation for (a) 616 bonds, limit* $= 0.02$*, (b) 606 bonds, limit* $= 0.03$

4.3 Discussion

The two algorithms used, namely Algorithm 3 (edge approximation) and Algorithm 4 (control area), gave approximately the same value of $\bar{p_c^b}$. Therefore we can assume that both algorithms give the same result. Algorithm 3 is simpler and runs faster than Algorithm 4. It is also a better one in the sense that it does not reduce the size of the networks, thus making it a better choice. The algorithms used are, roughly speaking, the same as those used for finding p_c^s for site percolation. The only adaptation is that now we consider bonds rather than sites. For a given Voronoi tessellation there are more bonds than sites, hence the correctness of the result should be better than that of site percolation.

The p_c^b from the total of 75 simulations was found to be 0.658. The higher value of p_c could have been predicted simply by noticing that, for a Voronoi network, the arrangement of bonds gives higher tendency of forming side branches than the arrangement of sites. These side branches can be considered as redundancies because they do not contribute directly to the percolating path, hence the expected higher value of p_c.

The plots of percentage of clusters of a specified size against p have the same characteristics as those obtained for site percolation (see bottom left of Figures 4.1*(a)* - 4.9*(b)* and 4.26*(a)* - 4.34*(b)* for bond percolation, and 3.1*(a)* - 3.9*(b)*, 3.12*(b)*, and 3.14*(a) and (b)* for site percolation), showing that the nature of the two problems, the site and the bond percolation, are the same. Phase inversion occurs at p_c^b and can be shown by the graph of maximum cluster sizes plotted against p (bottom right of Figures 4.1*(a)* - 4.9*(b)* and 4.26*(a)* - 4.34*(b)*). The implication of the phase inversion is that the network changes from being predominantly filled with occupied(vacant) bonds, to being predominantly filled with vacant(occupied) bonds. The p_c^b can be considered as the critical probability of percolation of blocked bonds (considering networks of tubes) as well as the critical probability of percolation of unblocked bonds (considering networks of blocked bonds).

Simulation results from smaller networks show higher variation in the value of p_c^b than those obtained from larger networks. This is due to the fact that the problem of percolation is a statistical one, hence larger sample size gives rise to higher accuracy. One would expect the average p_c^b ($\bar{p_c^b}$) obtained from several simulations on small networks and the p_c^b obtained from

a single run on a large network to be the same. This is confirmed, for the site percolation, by Figure 6.1, and for the bond percolation, by Figure 6.2 in Chapter 6.

Chapter 5

Bond percolation of some standard uniform lattices

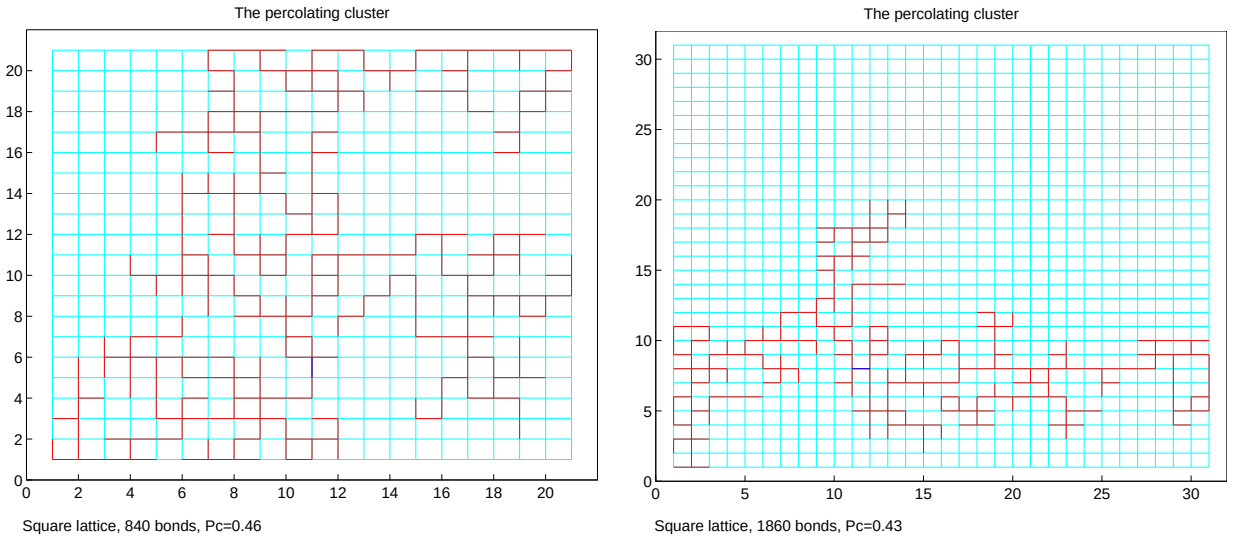
In order to prove that the algorithm used in obtaining the bond critical probability for Voronoi tessellations works correctly, the algorithm was used to find bond critical probability for some standard uniform lattices. The results obtained from simulations on square (4^4), triangular (3^6), hexagonal (6^3) and Kagomé (3.6.3.6) lattices, show close agreement with the values of critical probabilities obtained from the literature [42, 10, 45].

5.1 Square lattice

From 7 simulations on square lattices having number of bonds between 200 and 2000, the mean critical probability obtained is $\bar{p_c^{b,sq}} = 0.47$ and the variance is $\sigma^2_{p_c^{b,sq}} = 0.002$. The values obtained from literature survey are $0.493 - 0.498$ by Monte Carlo method [42] and 0.5000 by series method (exact) [10, 42]. The result obtained is 6% less than the exact value.

5.2 Triangular lattice

From 5 simulations on triangular lattices having number of bonds within the range 300-3000 bonds, the mean critical probability obtained is $\bar{p_c^{b,tr}} = 0.34$ and the variance is $\sigma^2_{p_c^{b,tr}} = 0.005$. The values obtained from literature survey are $0.341 - 0.349$ by Monte Carlo method [42] and $2\sin\left(\frac{\pi}{18}\right) = 0.3473$ by series method (exact) [10, 42]. The result obtained is 2% less than the



840 bonds,$p_c^{b,sq} = 0.46$ *1860 bonds*,$p_c^{b,sq} = 0.43$

Figure 5.1: *Bond percolation for square lattices of sizes 840 bonds and 1860 bonds*

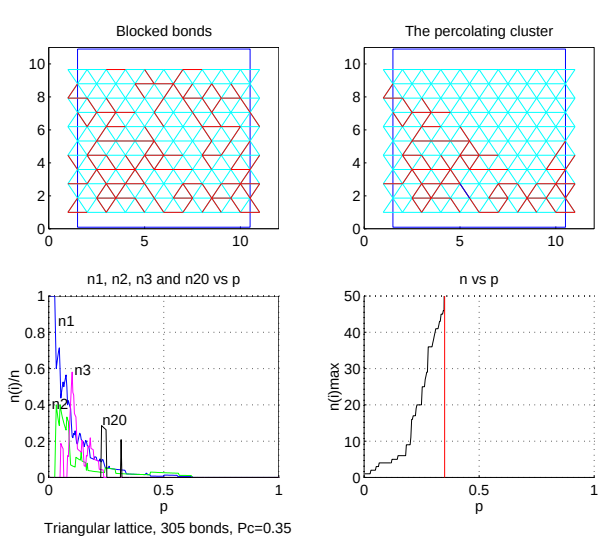
exact value.

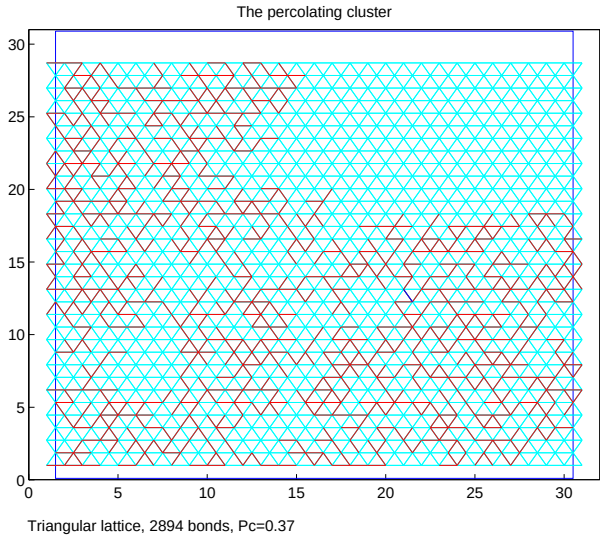
5.3 Honey comb lattice

From 3 simulations on honeycomb lattices having number of bonds within the range from 100 to 3000 bonds, the mean critical probability obtained is $\bar{p_c^{b,hc}} = 0.64$ and the variance is $\sigma^2_{\bar{p_c^{b,hc}}} = 0.0004$. The values obtained from literature survey are 0.640 by Monte Carlo method [42] and $1 - 2\sin\left(\frac{\pi}{18}\right) = 0.6527$ by series method (exact) [10, 42]. The result obtained is 2% less than the exact value.

5.4 Kagomé lattice

From 8 simulations on Kagomé lattices having number of bonds within the range from 300 to 3000 bonds, the mean critical probability is obtained $\bar{p_c^{b,Ka}} = 0.52$ and the variance is $\sigma^2_{\bar{p_c^{b,Ka}}} = 0.002$. The values obtained from literature survey are 0.435 by Monte Carlo method [42] and 0.525 [10]. The result obtained is 20% more than that cited by Shante and Kirkpatrick [42] and is 1% less than that given by Clerc et al. [10]





305 bonds,$p_c^{b,tr} = 0.35$ *2894 bonds*,$p_c^{b,tr} = 0.37$

Figure 5.2: *Bond percolation for triangular lattices of sizes 305 bonds and 2894 bonds*

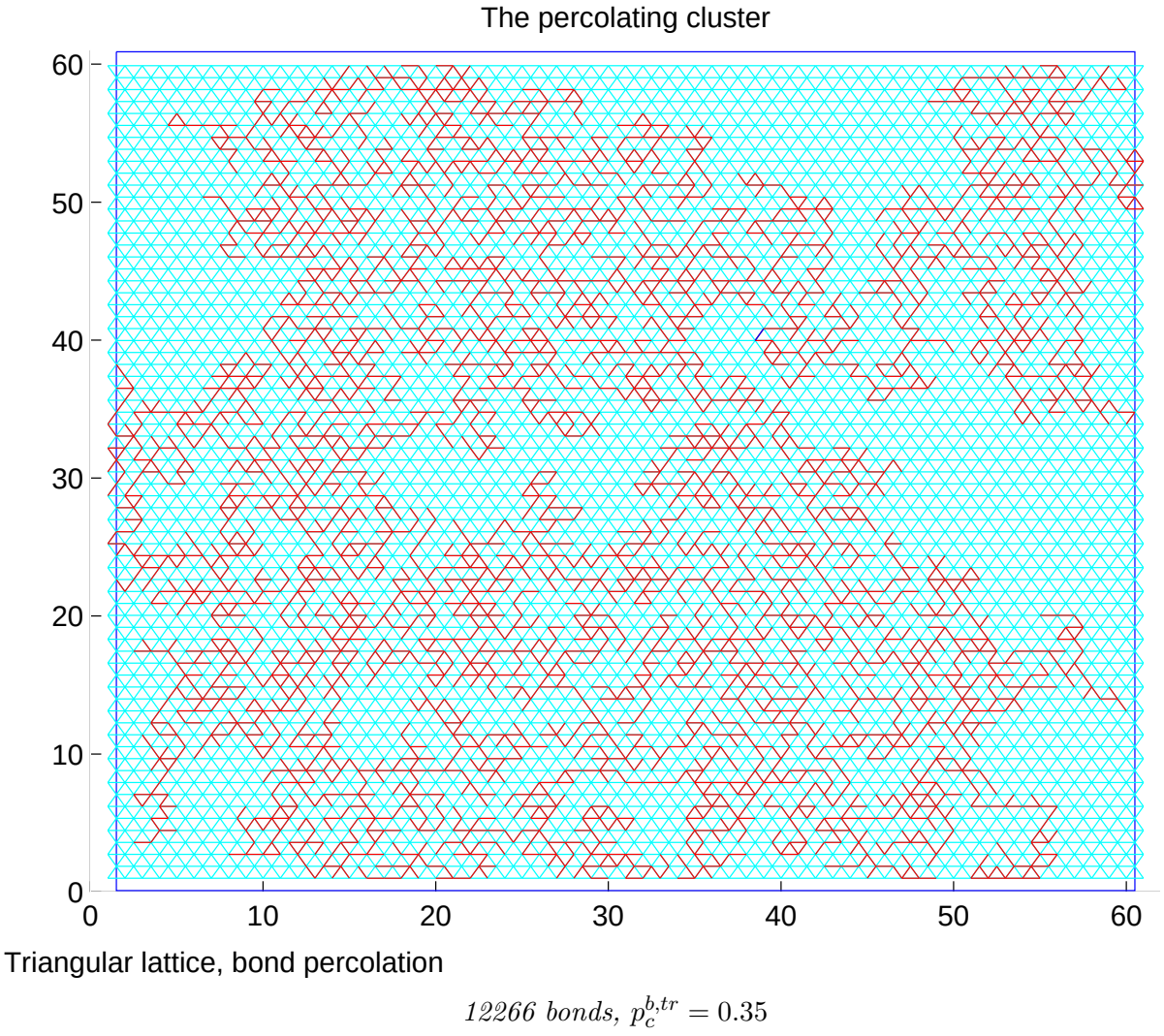
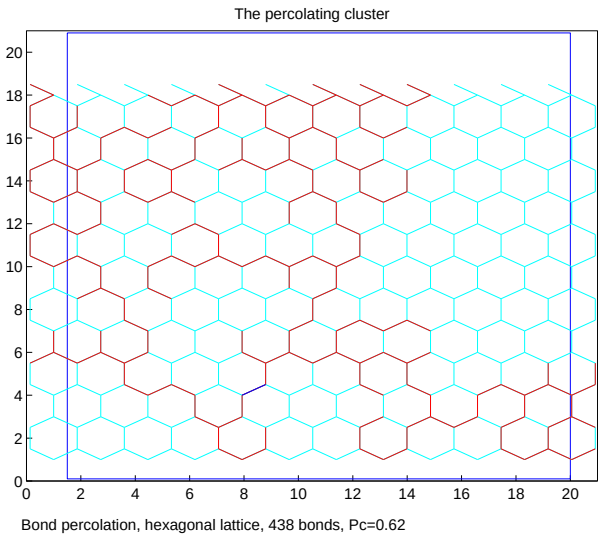
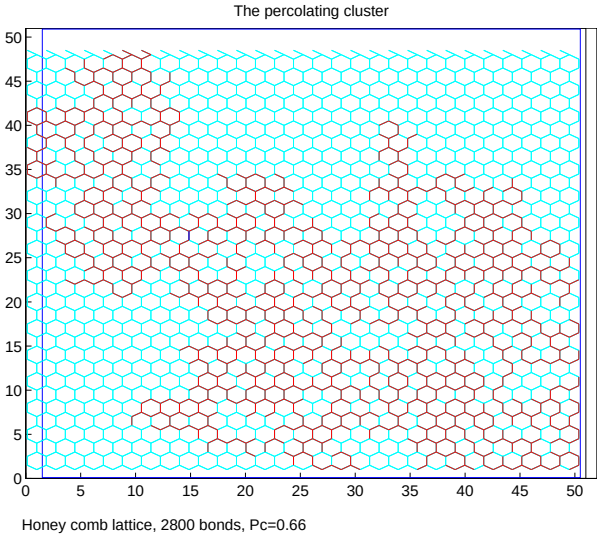


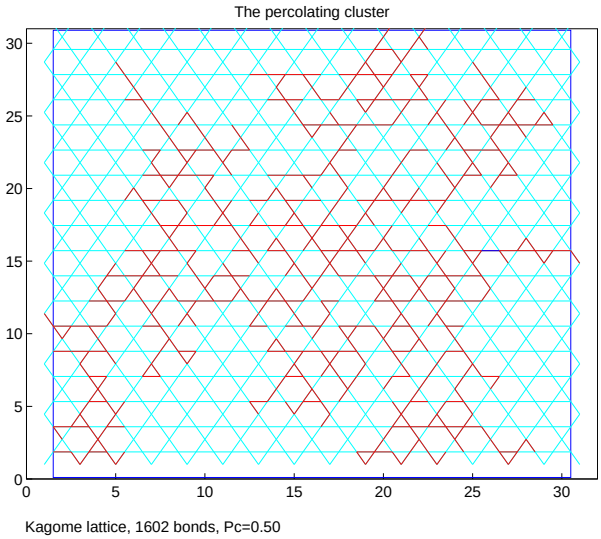
Figure 5.3: *Bond percolation for triangular lattices of size 12266 bonds*

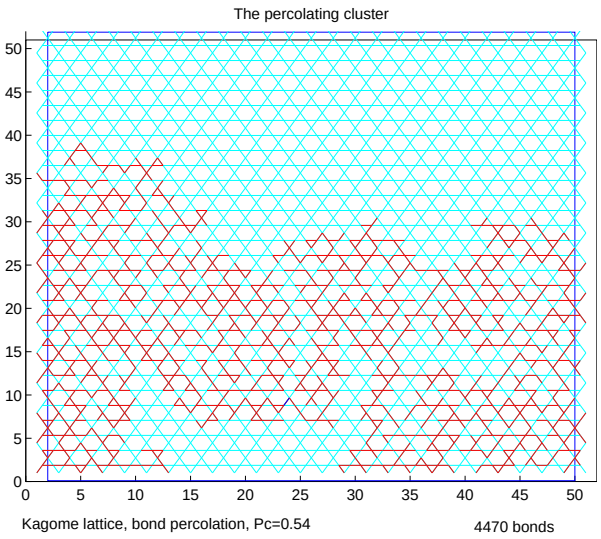




305 bonds,$p_c^{b,hc} = 0.62$ *2800 bonds*,$p_c^{b,hc} = 0.66$

Figure 5.4: *Bond percolation for honeycomb lattices of sizes 438 bonds and 2800 bonds*





1602 bonds,$p_c^{b,Ka} = 0.50$ *4470 bonds*,$p_c^{b,Ka} = 0.54$

Figure 5.5: *Bond percolation for Kagomé lattices of sizes 1602 bonds and 4470 bonds*

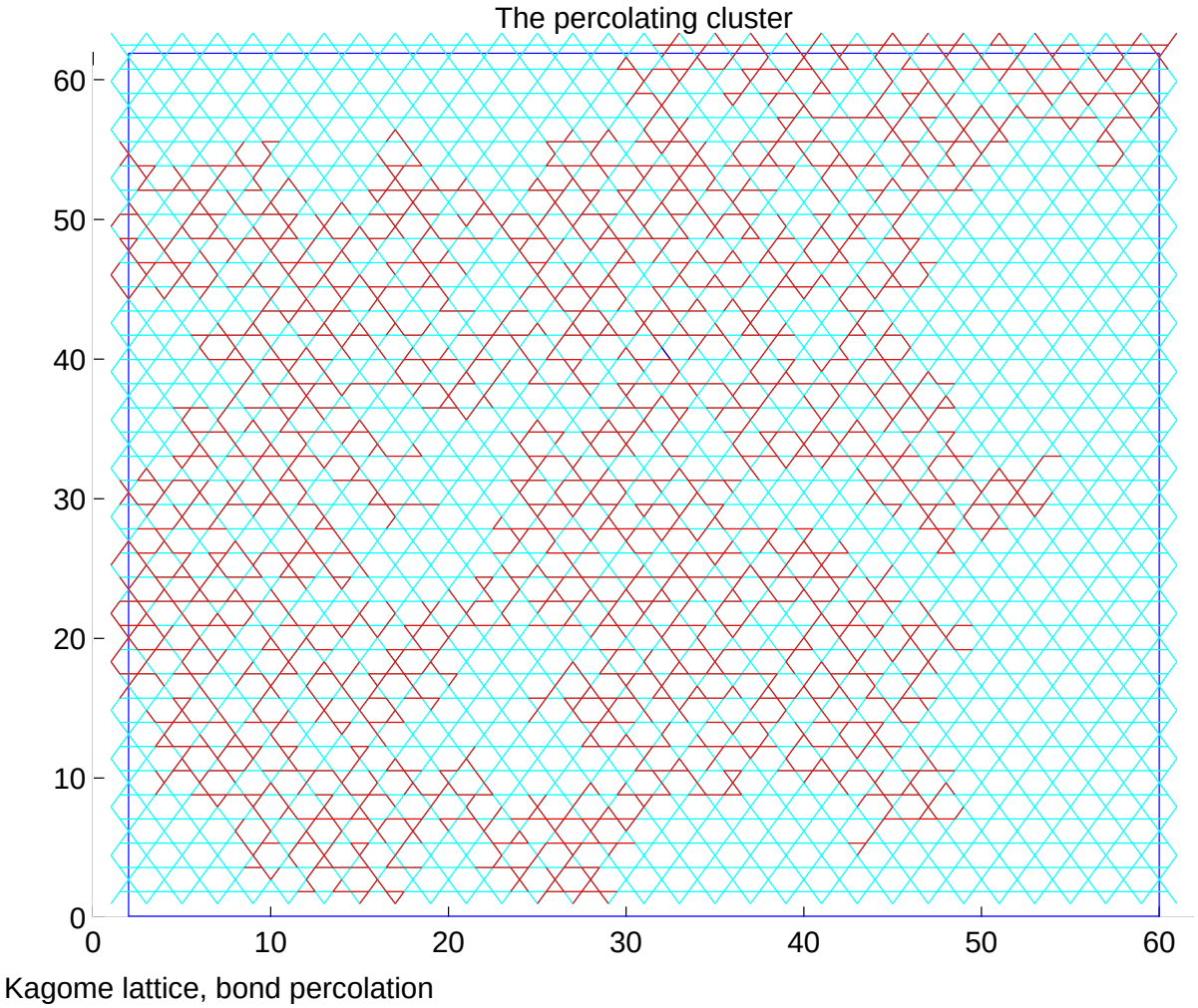


Figure 5.6: *Bond percolation for a Kagomé lattice of size 6444 bonds*

5.5 Discussion

By applying the Algorithm 3 to four uniform lattices, namely square lattice, triangular lattice, honeycomb lattice, and Kagomé lattice, the value of $\bar{p_c^b}$'s for the four lattices are obtained. These values ($p_c^{b,sq} = 0.47$, $p_c^{b,tr} = 0.34$, $p_c^{b,hc} = 0.64$, and $p_c^{b,Ka} = 0.52$) correspond closely to the values from the literature [10, 42, 45]. Therefore, we can assume that the Algorithm 3 used in finding $p_c^{b,Voronoi}$ works correctly, thus confirming the value $p_c^{b,Voronoi} = 0.658$ obtained.

The results obtained from simulations on uniform lattices can be applied to many natural structures because many patterns found in nature are very close to being uniform patterns. As an example, the surface of an ulcerated human stomach was found by using SEM to be close to a honeycomb lattice [13].

As an example of applications for the results from these simulations, consider a porous medium assumed to be a uniform lattice in structure, for example a triangular lattice in Figure 5.3. When there is a pressure difference in the fluid on either side of the structure, then the value $p_c^{b,tr} = 0.35$ represents the critical probability at which a percolation through the media occurs.

Chapter 6

Discussion

From a total number of 27 site percolation simulations of Voronoi networks of sizes between 70 and 500 cells, $\bar{p_c^s} = 0.507$. Figure 6.1 shows the mean values of p_c^s's from networks of different sizes as compared to the line $p_c^s = 0.507$. From the graph we can see that the results obtained are very consistant even though the sizes of networks used are small. The value of p_c^s does not change with increasing sizes of the networks, thus it represents one property intrinsic to a Voronoi network.

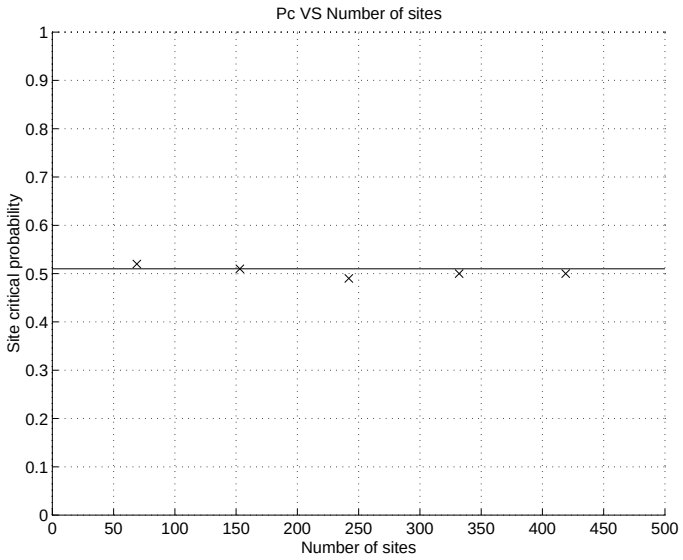


Figure 6.1: *$\bar{p_c^s}$'s plotted against number of sites*

From a total number of 75 simulations of Voronoi networks having sizes within the range of 100 to 8600 bonds, $\bar{p_c^b}$ is found to be 0.658. Figure 6.2 shows the mean values of p_c^b's obtained

from networks of different sizes in comparison with the overall mean value, which is shown as a line $p_c^b = 0.658$. The plot indicates that the value of the p_c^b obtained is very consistant. It also shows that p_c^b does not vary with increasing sizes of networks, therefore it can be used as one important parameter describing a Voronoi tessellation.

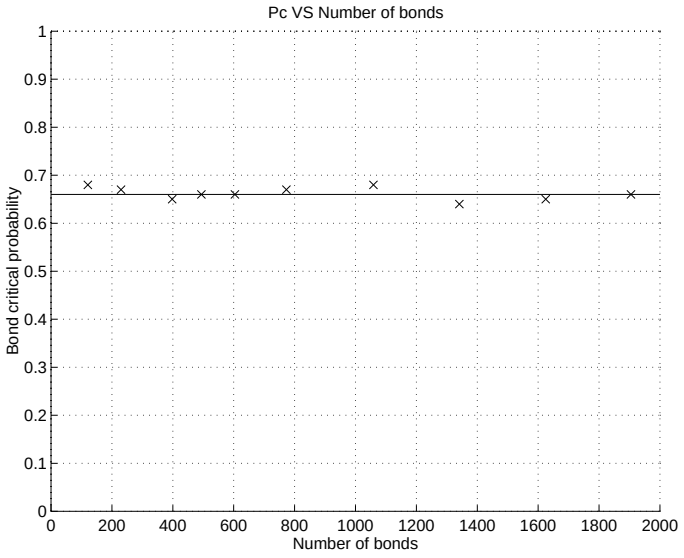


Figure 6.2: $\bar{p_c^b}$*'s plotted against number of bonds*

The variances of the critical probability for site percolation ($\sigma^2_{p_c^s}$) are plotted against sizes of networks in Figure 6.3. From the graph we can see that $\sigma^2_{p_c^s}$ becomes smaller as the networks gets larger.

For a bond problem we can draw a similar graph to Figure 6.3. This is shown in Figure 6.4. Again, the value of $\sigma^2_{p_c^b}$ also decreases with increasing sizes of networks. Both Figure 6.3 and Figure 6.4 tell us that if we want an accurate measurement of p_c we must either choose a very large network or do several runs on a smaller network and then use the average value of p_c obtained. The second approach is the one adopted here.

If we define a coordination number of a Voronoi tessellation as the average number of neighbours of sites (for site percolation) or bonds (for bond percolation), then Figure 6.5 shows that such a coordination number increases with network sizes starting from the value 0 for a network of size 1 and approaches 6 as the size increases. The coordination number (called the mean contact number by some authors) is theoretically 6 [24]. But efforts made in finding the value from a

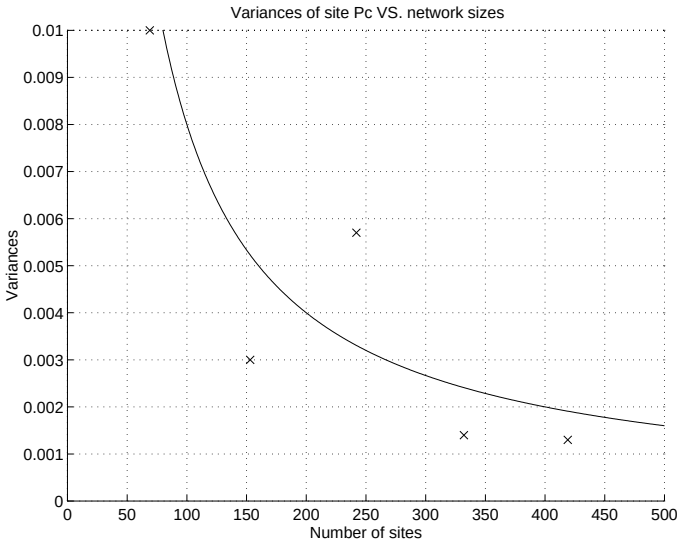


Figure 6.3: $\sigma^2_{p_c^s}$ *'s plotted against sizes of networks*

Voronoi tessellation tends to yield a value which is slightly less [34].

Again, we can draw a similar graph for a bond problem as shown in Figure 6.6. The figure shows that the value of the coordination number for bonds increases from 0 for only one bond and approaches 4 as the size of the network gets bigger.

Let the coordination number be represented by z. If we plot a graph between p_c and $\frac{1}{z}$ we will get a curve similar to the one shown in Figure 6.7 [10].

From the graph, we can predict the value of p_c from the knowledge of the coordination number. Using the average number of neighbours as a coordination number for a Voronoi tessellation, one can predict the values of p_c^s and p_c^b from Figure 6.5, 6.6 and 6.7, and compare these with those values obtained from the simulations. The results are shown in Figure 6.8 and Figure 6.9.

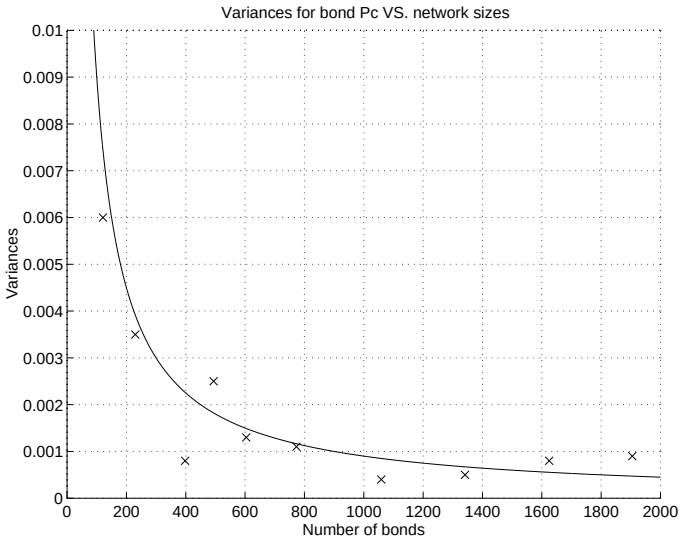


Figure 6.4: $\sigma^2_{p_c^b}$*'s plotted against sizes of networks*

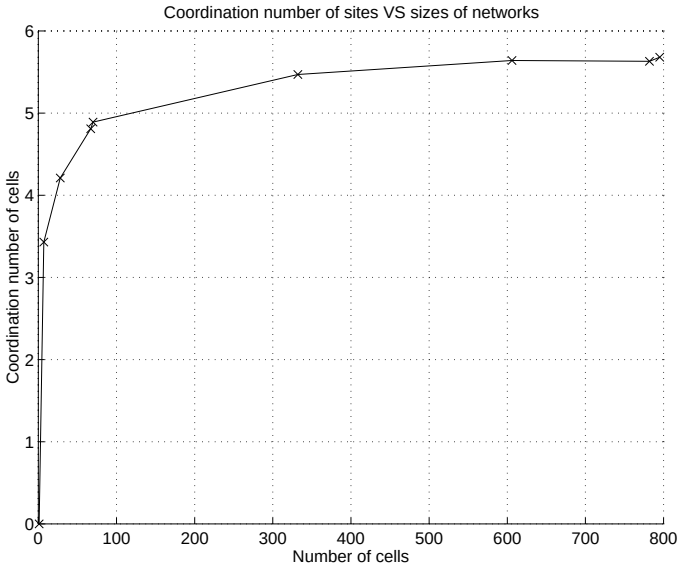


Figure 6.5: *Coordination number of sites plotted against network sizes*

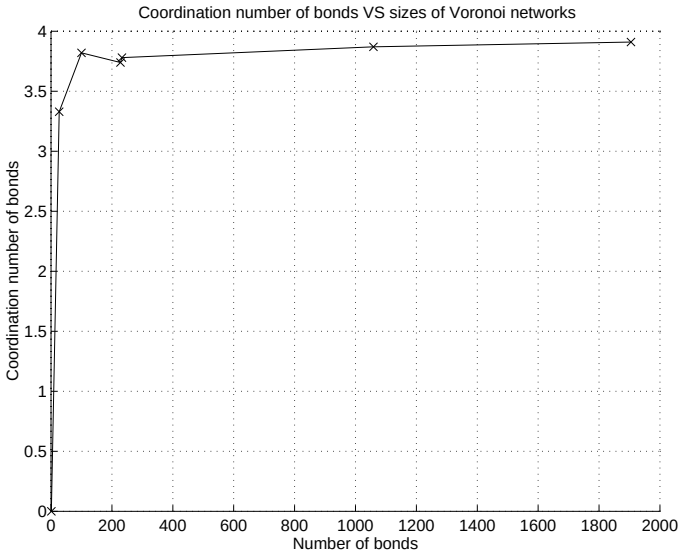


Figure 6.6: *Coordination number of bonds plotted against network sizes*

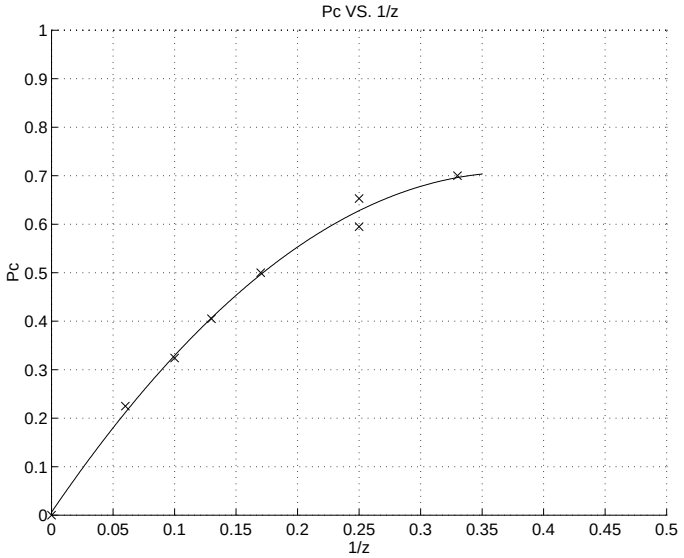


Figure 6.7: p_c *vs.* $\frac{1}{z}$ *for lattices in general*

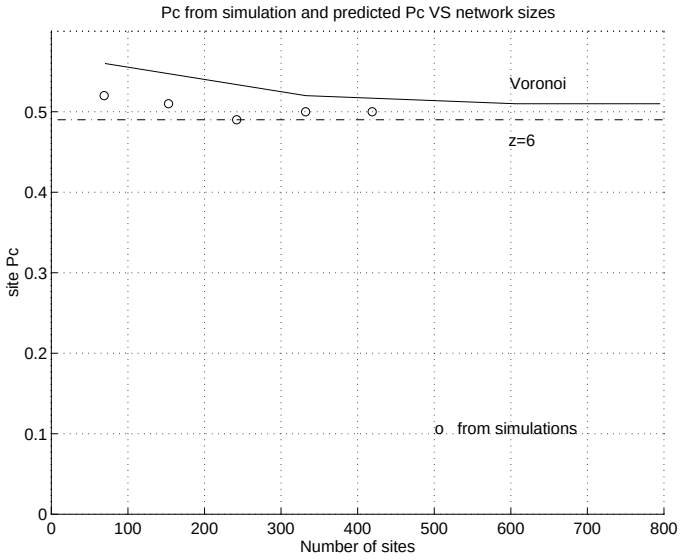


Figure 6.8: *p_c^s's from prediction compared with those ones from simulations*

Both Figure 6.8 and Figure 6.9 show that it is possible to guess the value of p_c^s and p_c^b of a Voronoi network without doing any simulation simply by finding the average number of neighbours. The computing time needed for this is much less than that required for a percolation simulation. So this can be used as an alternative approach in estimating the value of p_c.

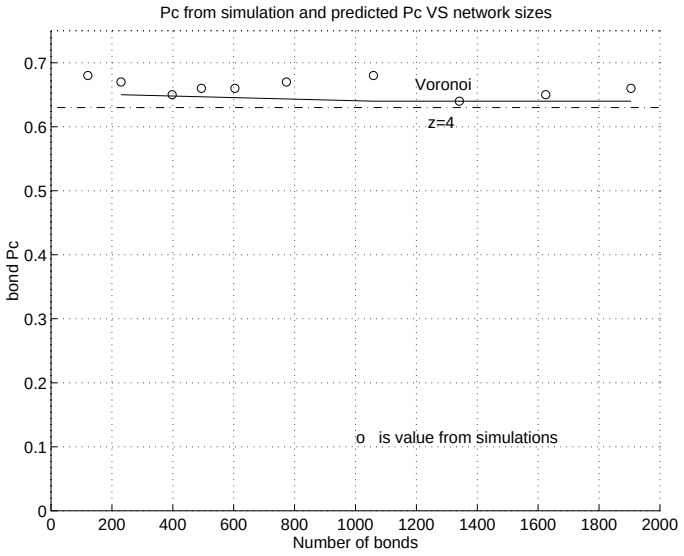


Figure 6.9: *p_c^b's from prediction compared with those ones from simulations*

Chapter 7

Conclusions

The contributions made in this dissertation are the following.

1. Algorithms for site percolation simulations (algorithm 1 and algorithm 2) and programs.
2. Algorithms for bond percolation simulations (algorithm 3 and algorithm 4) and programs.
3. Programs for generating square lattice, triangular lattice, honeycomb lattice, and Kagomé lattice.
4. A simulated value of $p_c^{b,Voronoi}$ ($p_c^{b,Voronoi} = 0.658$).

A survey has been made of the literature on Voronoi tessellation, percolation and tilings. Values of p_c^s and p_c^b for a Voronoi tessellation were found by computer simulations. The value obtained from the simulations indicates that $p_c^s \approx 0.507$ for a Voronoi tessellation. For bond percolation, the p_c^b was found to be ≈ 0.658.

In order to confirm that the algorithm works correctly, it was tried on several uniform lattices with known values of p_c^b's and the results obtained agree closely to those given by many authors. From simulations, $p_c^{b,sq} = 0.47$, $p_c^{b,tr} = 0.34$, $p_c^{b,hc} = 0.64$, and $p_c^{b,Ka} = 0.52$. The exact values of $p_c^{b,sq}$ and $p_c^{b,hc}$ are $p_c^{b,sq} = 0.5000$ and $p_c^{b,hc} = 0.6527$, which are in good agreement with the results obtained from the simulations. The value of $p_c^{b,tr}$ and $p_c^{b,Ka}$ simulated also correspond closely to those values found in literature [10, 42, 45]. Also the value $p_c^b = 0.658$ can be predicted from the relationship between p_c^b and z of networks in general. Thus it can be concluded that the result obtained for a Voronoi tessellation is also correct. Figure 7.1 is a plot between number of bonds

and number of sites in a Voronoi tessellation obtained from the networks used for simulations. The graph is approximately a straight line passing through the origin and having a slope of 3.1.

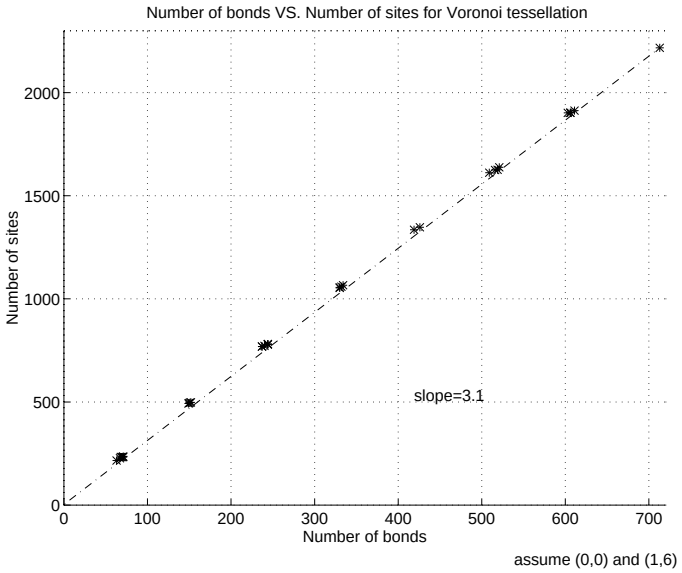


Figure 7.1: *Number of bonds as a function of number of sites for Voronoi tessellations*

Figure 7.2 shows all the average p_c's obtained from simulations on different types of lattices.

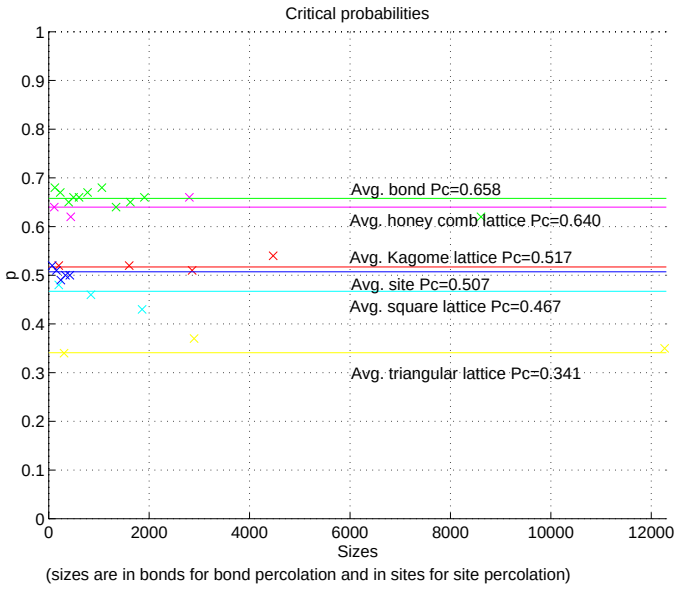


Figure 7.2: *p_c's bond problem, site problem, square, triangular, honeycomb, and Kagomé lattices*

The main applications of this work can be summarized as follows.

1. Bond percolation can be applied to flow of fluids through porous media [16, 40].

2. Site percolation can be applied to permeation through filtration membranes [25, 27, 53, 31].

7.1 Future work

There is much work still to be done in this area of research. For example,

i. **For 2-D networks**

Find p_c^s and p_c^b for other uniform lattices, as well as for Voronoi tessellations both with and without modification.

ii. **For 3-D networks**

Find p_c^s and p_c^b for 3-D uniform lattices and 3-D Voronoi networks.

iii. p_c^s **and** p_c^b

Find the relationship between the critical probability and various network statistics, for example we have seen that the values of p_c^s and p_c^b both depends upon z.

iv. **Network statistics**

Find network statistics, for example average length of edges, average cells volume, number of edges per cell, etc. [25, 27]

Appendix A

Algorithms

The following are all algorithms relevant to the simulations done in this project.

A.1 Algorithms used to prepare networks

There are essentially 3 steps in the generating of a Voronoi tessellation, they are

i. Choose a distribution of nuclei and specify the boundary conditions

ii. Create nuclei

iii. Create the Voronoi tessellation according to the Definition 1.1

V. Icke and R. van de Weygaert give a definitive description on the algorithm they used in generating the Voronoi tessellation in 2-dimension [24].

A.2 Algorithms used to find percolation

A.2.1 Site percolation

For site percolation two different algorithms were used. The first one takes into account next-nearest neighbours as well as nearest neighbours, while the second one only takes into account nearest neighbours. These are described in Algorithm 1 and Algorithm 2 respectively. Both of them gives similar results for Voronoi networks.

Algorithm 1 *Algorithm for site percolation using a next-nearest neighbourhood criteria.*

Prepare data files *There are 3 files. The first one contains x- and y-coordinates of all the nuclei. The second one contains x- and y-coordinates of vertices of each cell. The third one contains indices linking the first and the second files together.*

Find neighbours and find cell span *For each cell, find all the* **next-nearest neighbours** *of it. This is done by comparing the coordinates of vertices of each cell to those of the remaining cells. Also, for each cell, find its span in x-direction.*

for $i = 1$ *to the last cell*
  for $j = 1$ *to the last vertice belonging to the cell i*
    for $k = (i + 1)$ *to the last cell*
      for $l = 1$ *to the last vertice belonging to the cell k*
        if *the vertice j and vertice l are the same point* **then**
          cell i and cell k are neighbours
        end if
      end for
    end for
    find the smallest and the biggest x-coordinates of all vertices belonging to the cell i
  end for
  record the values of the smallest and the biggest x-coordinates of all vertices belonging to the cell i
end for

Find p_c^s *Randomly choose a new vacant cell. Update the statistics of clusters of neighbouring cells as more cells are added, these are number of clusters and the list of all the members in them. When the span of one cluster in x-direction covers the width of the networks calculate the value of* p_c^s *from the amount of occupied cells and the total cell amount.*

```
define the percolation distance
TempClusters ← ∅
Percolate ← false
while Percolate = false do
    Clusters ← ∅
    choose a cell randomly
    the first cluster of Clusters ← the cell
    Has the cell been chosen before this ?
    if it has not then
        for i = 1 to the last cluster of TempClusters
            Is the cell either a neighbour or a next-nearest neighbour to any
            of the cells in the cluster i of TempClusters ?
            if it is then
                the first cluster of Clusters ← the cluster i of TempClusters
            else
                a new cluster of Clusters ← the cluster i of TempClusters
            end if
            if Percolate = false then
                for i = 1 to the last cluster of Clusters
                    find the span in x-axis of the cluster i
                    if the span covers the percolation distance then
                        Percolate ← true
                        find the critical probability
                    end if
                end for
            end if
        end for
        TempClusters ← ∅
        TempClusters ← Clusters
    end if
end while
```

Algorithm 2 *Algorithm for site percolation using a nearest neighbourhood criteria.*

Prepare data files *There are 3 files. The first one contains x- and y-coordinates of all the nuclei. The second one contains x- and y-coordinates of vertices of each cell. The third one contains indices linking the first and the second files together.*

Find neighbours, find the left-most and the right-most cells *For each cell, find all the* **nearest neighbours** *of it. First, number all the vertices in the networks. Then find all the edges of each cell, which are nothing but pairs of numbered vertices. Two cells are nearest neighbours to each other if they have an edge in common. Find cells which have at least one of their vertices outside the control area.*

```
number all the vertices
find all the bonds, number them, record the two vertices each of the bonds connects,
and record all the cells which they belong to
for i = 1 to the last cell
    for j = (i + 1) to the last cell
        if both cell i and cell j have a bond in common then
            they are nearest neighbours
        end if
    end for
end for
define the percolation range on x-axis, ie the range which percolation is implies once it is traversed
for every cell
    for every vertices in that cell
        find out if there is any of the vertices of the cell which has its x-coordinate less than
        the lower limit of the percolation range
        find out if there is any of the vertices of the cell has its x-coordinate greater than
        the higher limit of the percolation range
    end for
    if there exists a vertice belonging to the cell which has an x-coordinate less than
    the lower limit of the percolation range then
        the cell is one of the left-most cells
    end if
    if there exists a vertice belonging to the cell which has an x-coordinate greater than
```

the higher limit of the percolation range **then**
the cell is one of the right-most cells
end if
end for

Find p_c^s *Randomly choose a new vacant cell. Update the statistics of clusters of neighbouring cells as more cells are being added. When there exists a single cluster which has got both a left-most and a right-most cells, calculate the value of* p_c^s *from the amount of occupied cells and the total cell amount.*

TempClusters $\leftarrow \emptyset$
Percolate $\leftarrow$ *false*
while *Percolate* = *false* **do**
Clusters $\leftarrow \emptyset$
choose a cell randomly
the first cluster of Clusters $\leftarrow$ *the cell*
Has the cell been chosen before this ?
if *it has not* **then**
for $i = 1$ *to the last cluster of TempClusters*
Is the cell either a neighbour or a next-nearest neighbour to any of the cells in the cluster i *of TempClusters ?*
if *it is* **then**
the first cluster of Clusters $\leftarrow$ *the cluster* i *of TempClusters*
else
a new cluster of Clusters $\leftarrow$ *the cluster* i *of TempClusters*
end if
for *all clusters of Clusters*
if *there exists any of the left-most cells within the cluster* **then**
if *there exists any of the right-most cells within the cluster* **then**
if *Percolate* = *false* **then**
Percolate = *true*
find the critical probability
end if
end if
end if
end for
end for
TempClusters $\leftarrow \emptyset$
TempClusters $\leftarrow$ *Clusters*
end if
end while

A.2.2 Bond percolation

In doing bond percolation, 2 different algorithms were implemented (Algorithm 3 and Algorithm 4). Both of them uses the idea of control area to avoid dealing with ragged edges of the Voronoi networks. The first one takes into account all the bonds within the networks while the second one omits those which are outside the control area. The descriptions of both algorithms can be found below. Both gives similar results, so I tend to use the first one more often since it does not reduce the size of the networks.

Algorithm 3 *Algorithm for bond percolation using the whole Voronoi networks created.*

Prepare data files *There are 3 files. The first one contains* x- *and* y-*coordinates of all the nuclei. The second one contains* x- *and* y-*coordinates of vertices of each cell. The third one contains indices linking the first and the second files together.*

Find neighbours *For each bond, find all the* **neighbours** *of it. First, number all the vertices in the networks. Find all the bonds, and then find all the neighbours of every bond by testing whether they have one end in common.*

number all the vertices
number all the bonds
for $i = 1$ *to the last bond*
for $j = (i + 1)$ *to the last bond*
if *the coordinates of one of the ends of bond* i *coincides with those of one of the ends of bond* j **then**
bond i *and bond* j *are neighbours*
end if
end for
end for

Find left-most and right-most bonds *These are bonds which have at least one end outside the control area.*

```
define the percolation range on x-axis
for every bond
    if the x-coordinate of any of the two ends of the bond is less than the lower limit of
    the percolation range then
        that bond is one of the left-most bonds
    end if
    if the x-coordinate of any of the two ends of the bond is greater than the higher limit of
    the percolation range then
        that bond is one of the right-most bonds
    end if
end for
```

Find p_c^s *Randomly choose a new vacant cell from all the cells that are within the control area. Update the statistics of clusters of neighbouring bonds as more cells are being added. When there is one cluster which has got both a left-most and a right-most bonds, calculate the value of* p_c^s *by dividing the amount of occupied bonds by the total amount of bonds.*

```
TempClusters ← ∅
Percolate ← false
while Percolate = false do
    Clusters ← ∅
    from all the bonds, choose one randomly
    the first cluster of Clusters ← the bond
    Has the bond ever been chosen before ?
    if it has not then
        for i = 1 to the last cluster of TempClusters
            Is the bond a neighbour of any of the bonds in the cluster i of TempClusters ?
            if it is then
                the first cluster of Clusters ← the cluster i of TempClusters
            else
                a new cluster of Clusters ← the cluster i of TempClusters
            end if
            for all clusters of Clusters
                if there exists any of the left-most bonds within the cluster then
                    if there exists any of the right-most bonds within the cluster then
                        if Percolate = falsethen
                            Percolate = true
                            find the critical probability
                        end if
                    end if
                end if
            end for
        end for
        TempClusters ← ∅
        TempClusters ← Clusters
    end if
end while
```

Algorithm 4 *Algorithm for bond percolation which discard bonds which are not within a specified control area.*

Prepare data files *There are 3 files. The first one contains x- and y-coordinates of all the nuclei. The second one contains x- and y-coordinates of vertices of each cell. The third one contains indices linking the first and the second files together.*

Find neighbours *For each bond, find all the* **neighbours** *of it by; first, numbering all the vertices in the networks. Then find all the bonds and see whether they have one end in common.*

Find left-most and right-most bonds *These are bonds which have at least one end outside the control area.*

Find p_c^s *Randomly choose a new vacant cell. Update the statistics of clusters of neighbouring bonds as more cells are being added. When there is one cluster which has got both a left-most and a right-most bonds, calculate the value of* p_c^s *by dividing the amount of occupied bonds by the total amount of bonds.*

```
define the control area, in other words the area of interest within the networks
define the percolation range as the x-dimension, in other words the width, of the control area
find all the bonds that are within the area
TempClusters ← ∅
Percolate ← false
while Percolate = false do
    Clusters ← ∅
    from all the bonds, choose one randomly
```

```
        while the bond is outside the control area do
            choose another bond randomly
        end while
        the first cluster of Clusters ← the bond
        Has the bond ever been chosen before ?
        if it has not then
            for i = 1 to the last cluster of TempClusters
                Is the bond a neighbour of any of the bonds in the cluster i of TempClusters ?
                if it is then
                    the first cluster of Clusters ← the cluster i of TempClusters
                else
                    a new cluster of Clusters ← the cluster i of TempClusters
                end if
                for all clusters of Clusters
                    if there exists any of the left-most bonds within the cluster then
                        if there exists any of the right-most bonds within the cluster then
                            if Percolate = falsethen
                                Percolate = true
                                find the critical probability
                            end if
                        end if
                    end if
                end for
            end for
            TempClusters ← ∅
            TempClusters ← Clusters
        end if
    end while
```

Appendix B

Programs

B.1 Programs used to prepare networks

These are *points.c* and *voronoi_plane.c* which are written by Dr. R. Jafferali and can be found in his thesis for Ph.D. [27], so they will not be described in detail here except that, *points.c* creates nuclei and *voronoi_plane.c* creates a Voronoi tessellation.

B.2 Programs used to find percolation

There are 4 programs. They correspond to the 4 algorithms mentioned earlier in Appendix 1. All the programs are written in MATLAB. I found that it is easier to write a simulation program on MATLAB than to do it in C. Nevertheless, the program corresponding to Algorithm 3 has been converted to C, and the result was used in several runs though it does not run any faster than the one written for MATLAB. I have included both the one in C and the one in MATLAB here.

B.2.1 Site percolation

B.2.1.1 Program written from a next-nearest neighbours algorithm

This program follows the Algorithm 1. It was used for producing Figures 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, 3.13.

```
clear;                          clf;
LowerValueFrame =0.05;          HigherValueFrame =0.95;
PtrPPyIdx =fopen('./PPyIdx3000L01','r');
Temp =fscanf(PtrPPyIdx,'%d %d');
for i=1:2:(size(Temp)-1),
   CellMap =[CellMap;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrPPyIdx);
CellAmount =size(CellMap,1);
PtrP =fopen('./P3000L01','r');
TempAmount =fscanf(PtrP,'%d',1);
Temp =fscanf(PtrP,'%f %f',inf);
for i=1:2:(size(Temp)-1),
   Nuclei =[Nuclei;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrP);
NucleiAmount =size(Nuclei,1);
PtrPoly =fopen('./Poly3000L01','r');
i =0;
while i<CellAmount,
   i =i+1;
   TempAmount =fscanf(PtrPoly,'%d',1);
   Cell =[Cell;TempAmount];
   Temp =fscanf(PtrPoly,'%f %f',(2*TempAmount));
   for j=1:2:(size(Temp)-1),
      Vertices =[Vertices;Temp(j,:),Temp((j+1),:)];
   end
end
fclose(PtrPoly);
```

```
Accumulate =Cell(1);
ForFill =[1,Accumulate];
for i=2:CellAmount,
   Accumulate =Accumulate +Cell(i);
   ForFill(i,:) =[(ForFill((i-1),2) +1),Accumulate];
end
Neighbours =sparse([],[],[],CellAmount,CellAmount,0);
for i=1:CellAmount,
   for j=ForFill(i,1):ForFill(i,2),
      for k=(i+1):CellAmount,
         for l=ForFill(k,1):ForFill(k,2),
            if ((Vertices(j,1)==Vertices(l,1))&(Vertices(j,2)==Vertices(l,2)))
               Neighbours(i,k) =1;
               Neighbours(k,i) =1;
            end
         end
      end
   end
   RimLeft =[RimLeft;min(Vertices(ForFill(i,1):ForFill(i,2),1))];
   RimRight =[RimRight;max(Vertices(ForFill(i,1):ForFill(i,2),1))];
end

TempClusters =sparse([],[],[],CellAmount,1,0)
Percolate =0;
while Percolate==0,
   BlockedCell =ceil(CellAmount*rand(1));
   clear BlockedMask;
   BlockedMask =sparse([BlockedCell],[1],[1],CellAmount,1,1);
   clear Clusters;
   Clusters =BlockedMask;
   SizeTempClusters =size(TempClusters,2);
   Overlapped =0;
   for i=1:SizeTempClusters,
      Overlapped =Overlapped + max(BlockedMask & TempClusters(:,i));
   end
   if Overlapped==0
      for i=1:SizeTempClusters,
         clear TouchCondition;
         TouchCondition =max(Neighbours(:,BlockedCell) & TempClusters(:,i));
         clear ClusterExist;
         ClusterExist =max(TempClusters(:,i));
         if ((TouchCondition==0) & (ClusterExist~=0))
            Clusters =[Clusters,TempClusters(:,i)];
         else
            Clusters(:,1) =Clusters(:,1) | TempClusters(:,i);
         end
      end
      ClustersAmount =size(Clusters,2);
      i =0;
      if Percolate==0
         while i < ClustersAmount,
            i =i+1;
            LeftEnd =min( RimLeft.*(Clusters(:,i)+10000*(~(Clusters(:,i)) ) ) );
            RightEnd =max(RimRight.*Clusters(:,i));
            if (( LeftEnd < LowerValueFrame) & (RightEnd > HigherValueFrame) )
               Percolate =1;
               PercolatingCluster =Clusters(:,i);
               KeyCell =BlockedCell;
               hold on;
               plot(Nuclei(:,1),Nuclei(:,2),'k.');
               Count = 1;
               Loop = size(Cell);
               for i=1:Loop;
                  SmallLoop = Cell(i,1);
          for j=Count:(Count+SmallLoop-2);
                        x = [Vertices(j,1) Vertices(j+1,1)];
                        y = [Vertices(j,2) Vertices(j+1,2)];
                        plot(x,y,'c');
```

```
                end
                x = [Vertices(Count,1) Vertices(j+1,1)];
                y = [Vertices(Count,2) Vertices(j+1,2)];
                plot(x,y,'c');
                Count = Count + SmallLoop;
             end
             plot([0 1 1],[1 1 0],'k');
             [IPercolatingCluster,JPercolatingCluster] =find(PercolatingCluster);
             for i=1:size(IPercolatingCluster,1),
                fill(Vertices(ForFill(IPercolatingCluster(i),1): ...
                ForFill(IPercolatingCluster(i),2),1),...
                Vertices(ForFill(IPercolatingCluster(i),1): ...
                ForFill(IPercolatingCluster(i),2),2),'r');
             end
                fill(Vertices(ForFill(KeyCell,1):ForFill(KeyCell,2),1),...
                Vertices(ForFill(KeyCell,1):ForFill(KeyCell,2),2),'b');
                plot([LowerValueFrame HigherValueFrame HigherValueFrame ...
                LowerValueFrame LowerValueFrame],...
                [0.98 0.98 0.02 0.02 0.98],'b');
                title('The percolating cluster');
                CriticalProb =sum(sum(Clusters))/CellAmount;
          end
       end
    end
    TempClusters =Clusters;
  end
end
```

B.2.1.2 Program written from a nearest neighbours algorithm

This program follows the Algorithm 2. It was used for producing Figure 3.14.

```
clear;                  clf;                  echo on;
LowerValueFrame =0.1;   HigherValueFrame  =0.9;
PtrPPyIdx =fopen('./PPyIdx800L01','r');
Temp =fscanf(PtrPPyIdx,'%d %d');
for i=1:2:(size(Temp)-1),
  CellMap =[CellMap;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrPPyIdx);
CellAmount =size(CellMap,1);
PtrP =fopen('./P800L01','r');
TempAmount =fscanf(PtrP,'%d',1);
Temp =fscanf(PtrP,'%f %f',inf);
for i=1:2:(size(Temp)-1),
  Nuclei =[Nuclei;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrP);
NucleiAmount =size(Nuclei,1);
PtrPoly =fopen('./Poly800L01','r');
i =0;
while i<CellAmount,
  i =i+1;
  TempAmount =fscanf(PtrPoly,'%d',1);
  Cell =[Cell;TempAmount];
  Temp =fscanf(PtrPoly,'%f %f',(2*TempAmount));
  for j=1:2:(size(Temp)-1),
     Vertices =[Vertices;Temp(j,:),Temp((j+1),:)];
  end
end
fclose(PtrPoly);
Accumulate =Cell(1);
ForFill =[1,Accumulate];
for i=2:CellAmount,
  Accumulate =Accumulate +Cell(i);
  ForFill(i,:) =[(ForFill((i-1),2) +1),Accumulate];
end
VerticeAmount =size(Vertices,1);
Flags =ones(VerticeAmount,1);
```

```
VerticeNumbered =zeros(VerticeAmount,1);
Number =0;
for i=1:VerticeAmount,
   if Flags(i,1)==1,
      Number =Number+1;
      VerticeNumbered(i,1) =Number;
      for j=(i+1):VerticeAmount,
         if Flags(j,1)==1,
            if (Vertices(i,:)-Vertices(j,:))==[0,0],
               VerticeNumbered(j,1) =Number;
               Flags(j,1) =0;
            end
         end
      end
   end
end
VerticeTrueAmount =max(VerticeNumbered);
VerticesSorted =zeros(VerticeTrueAmount,2);
for i=1:VerticeAmount,
   VerticesSorted(VerticeNumbered(i,1),:) =Vertices(i,:);
end
Start =0;
for i=1:CellAmount,
   for j=1:(Cell(i,1)-1),
      CellNVertice =[CellNVertice;i];
      BondsExcess =[BondsExcess;VerticeNumbered((Start+j),1),VerticeNumbered((Start+j+1),1)];
   end
   CellNVertice =[CellNVertice;i];
   BondsExcess =[BondsExcess;VerticeNumbered((Start+Cell(i,1)),1),VerticeNumbered((Start+1),1)];
   Start =Start+Cell(i,1);
end
BondsExcess =sort(BondsExcess')';
Neighbours =sparse([],[],[],CellAmount,CellAmount,0);
for i=1:VerticeAmount,
   for j=(i+1):VerticeAmount,
      if (BondsExcess(i,:)==BondsExcess(j,:))
         Neighbours(CellNVertice(i,1),CellNVertice(j,1)) =1;
         Neighbours(CellNVertice(j,1),CellNVertice(i,1)) =1;
      end
   end
end
LeftRim =sparse([],[],[],CellAmount,1,0);
RightRim =sparse([],[],[],CellAmount,1,0);
k =0;
for i=1:CellAmount,
   Left  =0;
   Right  =0;
   for j=1:(Cell(i,1)),
      k =k+1;
      if (Vertices(k,1) < LowerValueFrame)
         Left =1;
      end
      if (Vertices(k,1) > HigherValueFrame)
         Right =1;
      end
   end
   if (Left==1)
      LeftRim(i,1) =1;
   end
   if (Right==1)
      RightRim(i,1) =1;
   end
end
TempClusters =sparse([],[],[],CellAmount,1,0);
Percolate =0;
while Percolate==0,
   BlockedCell =ceil(CellAmount*rand(1));
   clear BlockedMask;
```

```
   BlockedMask =sparse([BlockedCell],[1],[1],CellAmount,1,1);
   clear Clusters;
   Clusters =BlockedMask;
   SizeTempClusters =size(TempClusters,2);
   Overlapped =0;
   for i=1:SizeTempClusters,
      Overlapped =Overlapped + max(BlockedMask & TempClusters(:,i));
   end
   if Overlapped==0
      for i=1:SizeTempClusters,
         clear TouchCondition;
         TouchCondition =max(Neighbours(:,BlockedCell) & TempClusters(:,i));
         clear ClusterExist;
         ClusterExist =max(TempClusters(:,i));
         if ((TouchCondition==0) & (ClusterExist~=0))
            Clusters =[Clusters,TempClusters(:,i)];
         else
            Clusters(:,1) =Clusters(:,1) | TempClusters(:,i);
         end
      end
      ClustersAmount =size(Clusters,2);
      for i=1:ClustersAmount,
         if (max(Clusters(:,i) & LeftRim) > 0)
            if (max(Clusters(:,i) & RightRim) > 0)
               if (Percolate==0)
                  Percolate =1;
                  PercolatingCluster =Clusters(:,i);
                  KeyCell =BlockedCell;
                  hold on;
                  plot(Nuclei(:,1),Nuclei(:,2),'k.');
                  Count = 1;
                  Loop = size(Cell);
                  for i=1:Loop;
             SmallLoop = Cell(i,1);
             for j=Count:(Count+SmallLoop-2);
        x = [Vertices(j,1) Vertices(j+1,1)];
        y = [Vertices(j,2) Vertices(j+1,2)];
        plot(x,y,'c');
             end
             x = [Vertices(Count,1) Vertices(j+1,1)];
             y = [Vertices(Count,2) Vertices(j+1,2)];
             plot(x,y,'c');
             Count = Count + SmallLoop;
                  end
                  plot([0 1 1],[1 1 0],'k');
                  [IPercolatingCluster,JPercolatingCluster] =find(PercolatingCluster);
                  for i=1:size(IPercolatingCluster,1),
                     fill(Vertices(ForFill(IPercolatingCluster(i),1):...
                     ForFill(IPercolatingCluster(i),2),1),...
                     Vertices(ForFill(IPercolatingCluster(i),1):...
                     ForFill(IPercolatingCluster(i),2),2),'r');
                  end
                  fill(Vertices(ForFill(KeyCell,1):ForFill(KeyCell,2),1),...
                  Vertices(ForFill(KeyCell,1):ForFill(KeyCell,2),2),'b');
                  plot([LowerValueFrame HigherValueFrame HigherValueFrame LowerValueFrame ...
                  LowerValueFrame],[0.98 0.98 0.02 0.02 0.98],'b');
                  title('The percolating cluster');
                  CriticalProb =sum(sum(Clusters))/CellAmount;
               end
            end
         end
      end
      clear TempClusters;
      TempClusters =Clusters;
   end
end
CellAmount
text(-.1,-.1,'Site percolation, Voronoi tessellation, 1000 nuclei, limit=0.01');
```

```
print -depsc SiteVNr1000L01
```

B.2.2 Bond percolation

B.2.2.1 Programs written from an edge approximation algorithm

MATLAB program This program follows the Algorithm 3. It was used for producing Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.12, 4.14, 4.16, 4.17, 4.19, 4.20, 4.21, 4.23, 4.24.

```
clear;                      clf;               echo on;
LowerValueFrame   =0.05;   HigherValueFrame   =0.95;
PtrPPyIdx =fopen('./PPyIdx2000','r');
Temp =fscanf(PtrPPyIdx,'%d %d');
for i=1:2:(size(Temp)-1),
   CellMap =[CellMap;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrPPyIdx);
CellAmount =size(CellMap,1);
PtrP =fopen('./P2000','r');
TempAmount =fscanf(PtrP,'%d',1);
Temp =fscanf(PtrP,'%f %f',inf);
for i=1:2:(size(Temp)-1),
   Nuclei =[Nuclei;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrP);
NucleiAmount =size(Nuclei,1);
PtrPoly =fopen('./Poly2000','r');
i =0;
while i<CellAmount,
   i =i+1;
   TempAmount =fscanf(PtrPoly,'%d',1);
   Cell =[Cell;TempAmount];
   Temp =fscanf(PtrPoly,'%f %f',(2*TempAmount));
   for j=1:2:(size(Temp)-1),
      Vertices =[Vertices;Temp(j,:),Temp((j+1),:)];
   end
end
fclose(PtrPoly);
VerticeAmount =size(Vertices,1);
Flags =ones(VerticeAmount,1);
VerticeNumbered =zeros(VerticeAmount,1);
Number =0;
for i=1:VerticeAmount,
   if Flags(i,1)==1,
      Number =Number+1;
      VerticeNumbered(i,1) =Number;
      for j=(i+1):VerticeAmount,
         if Flags(j,1)==1,
            if (Vertices(i,:)-Vertices(j,:))==[0,0],
               VerticeNumbered(j,1) =Number;
               Flags(j,1) =0;
            end
         end
      end
   end
end
VerticeTrueAmount =max(VerticeNumbered);
VerticesSorted =zeros(VerticeTrueAmount,2);
for i=1:VerticeAmount,
   VerticesSorted(VerticeNumbered(i,1),:) =Vertices(i,:);
end
Start =0;
for i=1:CellAmount,
   for j=1:(Cell(i,1)-1),
      BondsExcess =[BondsExcess;VerticeNumbered((Start+j),1),VerticeNumbered((Start+j+1),1)];
   end
   BondsExcess =[BondsExcess;VerticeNumbered((Start+Cell(i,1)),1),VerticeNumbered((Start+1),1)];
   Start =Start+Cell(i,1);
end
```

```
BondsExcess =sort(BondsExcess')';
Flags =ones(size(BondsExcess,1),1);
for i=1:size(BondsExcess,1),
   for j=(i+1):size(BondsExcess,1),
      if (BondsExcess(i,:)-BondsExcess(j,:))==[0, 0],
         Flags(j,1) =0;
      end
   end
end
for i=1:size(BondsExcess,1),
   if Flags(i,1)==1
      Bonds =[Bonds;BondsExcess(i,:)];
   end
end
BondsAmount =size(Bonds,1);
Neighbours =sparse([],[],[],size(Bonds),size(Bonds),0);
for i=1:BondsAmount,
   for j=(i+1):size(Bonds,1),
      if (Bonds(i,1)==Bonds(j,1)) |(Bonds(i,1)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      elseif (Bonds(i,2)==Bonds(j,1)) |(Bonds(i,2)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      end
   end
end
LeftRim =sparse([],[],[],size(Bonds,1),1,0);
RightRim =sparse([],[],[],size(Bonds,1),1,0);
for i=1:BondsAmount,
   if (VerticesSorted(Bonds(i,1),1) < LowerValueFrame) | ...
      (VerticesSorted(Bonds(i,2),1) < LowerValueFrame)
      LeftRim(i,1) =1;
   end
   if (VerticesSorted(Bonds(i,1),1) > HigherValueFrame) | ...
      (VerticesSorted(Bonds(i,2),1) > HigherValueFrame)
      RightRim(i,1) =1;
   end
end
Percolate =0;
TempClusters =sparse([],[],[],BondsAmount,1,0);
while Percolate==0,
   BlockedBond =ceil(BondsAmount*rand(1));
   clear BlockedMask;
   BlockedMask =sparse([BlockedBond],[1],[1],BondsAmount,1,1);
   clear Clusters;
   Clusters =BlockedMask;
   SizeTempClusters =size(TempClusters,2);
   clear Overlapped;
   Overlapped =0;
   for i=1:SizeTempClusters,
      Overlapped =Overlapped  + max(BlockedMask & TempClusters(:,i));
   end
   if Overlapped==0
      for i=1:SizeTempClusters,
         clear TouchCondition;
         TouchCondition =max(Neighbours(:,BlockedBond) & TempClusters(:,i));
         clear ClusterExist;
         ClusterExist =max(TempClusters(:,i));
         if ((TouchCondition==0) & (ClusterExist~=0))
            Clusters =[Clusters,TempClusters(:,i)];
         else
            Clusters(:,1) =Clusters(:,1) | TempClusters(:,i);
         end
      end
      clear ClustersAmount;
      ClustersAmount =size(Clusters,2);
      for i=1:ClustersAmount,
```

```
        if (max(Clusters(:,i) & LeftRim) > 0)
           if (max(Clusters(:,i) & RightRim) > 0)
              if (Percolate==0)
                 Percolate =1;
                 PercolatingCluster =Clusters(:,i);
                 hold on;
                 plot(Nuclei(:,1),Nuclei(:,2),'k.');
                 Count = 1;
                 Loop = size(Cell);
                 for i=1:Loop;
                    SmallLoop = Cell(i,1);
                    for j=Count:(Count+SmallLoop-2);
                       x = [Vertices(j,1) Vertices(j+1,1)];
                       y = [Vertices(j,2) Vertices(j+1,2)];
                       plot(x,y,'c');
                    end
                    x = [Vertices(Count,1) Vertices(j+1,1)];
                    y = [Vertices(Count,2) Vertices(j+1,2)];
                    plot(x,y,'c');
                    Count = Count + SmallLoop;
                 end
                 plot([0 1 1],[1 1 0],'k');
                 [IPercolatingCluster,JPercolatingCluster]   =find(PercolatingCluster);
                 for i=1:size(IPercolatingCluster,1),
                    plot([VerticesSorted(Bonds(IPercolatingCluster(i),1),1),...
                    VerticesSorted(Bonds(IPercolatingCluster(i),2),1)],...
                    [VerticesSorted(Bonds(IPercolatingCluster(i),1),2),...
                    VerticesSorted(Bonds(IPercolatingCluster(i),2),2)],'r');
                 end
                 plot([VerticesSorted(Bonds(BlockedBond,1),1),...
                 VerticesSorted(Bonds(BlockedBond,2),1)],...
                 [VerticesSorted(Bonds(BlockedBond,1),2),...
                 VerticesSorted(Bonds(BlockedBond,2),2)],'b');
                 plot([LowerValueFrame HigherValueFrame HigherValueFrame...
                 LowerValueFrame LowerValueFrame],[0.98 0.98 0.02 0.02 0.98],'b');
                 title('The percolating cluster');
                 CriticalProb =sum(sum(Clusters))/BondsAmount
              end
           end
        end
     end
     clear TempClusters;
     TempClusters =Clusters;
  end
end
```

C program This program is a C version of the previous program. It was used for producing Figures 4.11, 4.13, 4.15, 4.18, 4.22.

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#define LowerValueFrame 0.2
#define HigherValueFrame 0.8
int    VerticesAmount,
       BondsExcessAmount,
       BondsAmount,
       LeftRimAmount,
       RightRimAmount,
       NucleiAmount,
       CellAmount,
       VerticesTrueAmount,
       NeighboursAmount;
int    *VerticesNumbered,
       *Cell,
       *LeftRim,
       *RightRim;
typedef struct {
       float    X;
       float    Y;
```

```
} Coordinates;
typedef struct {
       int      End1;
       int      End2;
} Ends;
typedef struct {
       int      Neighbour1;
       int      Neighbour2;
} Near;
typedef struct {
       int      MembersAmount;
       int      *Members;
}Group;
Coordinates    *Nuclei,
               *Vertices,
               *VerticesSorted;
Ends      *BondsExcess,
          *Bonds;
Near      *Neighbours;
void ReadFiles()
{
int     i,j,k,
        Temp1;
float   Temp2, Temp3;
FILE    *PtrP,
        *PtrPoly;
printf("ReadFile module ...\n");
if((PtrP =fopen("./P20","r"))==NULL){
   printf("Error cannot open the nuclei file for reading...\n");
   exit(0);
}
printf("reading nuclei file ...\n");
if ((Nuclei=malloc(2*sizeof(Coordinates)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
fscanf(PtrP,"%d",&Temp1);
NucleiAmount =0;
while(fscanf(PtrP,"%f %f",&Temp2,&Temp3)!=EOF) {
   Nuclei[++NucleiAmount].X =Temp2;
   Nuclei[NucleiAmount].Y =Temp3;
   Nuclei =realloc(Nuclei,(NucleiAmount+2)*sizeof(Coordinates));
}
if (fclose(PtrP))
   perror("Error closing file...\n");
if((PtrPoly =fopen("./Poly20","r"))==NULL){
   printf("Error cannot open the polygon file for reading...\n");
   exit(0);
}
printf("reading vertices file ...\n");
if ((Vertices=malloc(2*sizeof(Coordinates)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
if((Cell=malloc(2*sizeof(int)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
CellAmount =VerticesAmount  =0;
while(fscanf(PtrPoly,"%d",&Temp1)==1){
   Cell[++CellAmount] =Temp1;
   Cell =realloc(Cell,(CellAmount+2)*sizeof(int));
   for (j=1;j<=Cell[CellAmount];j++) {
      fscanf(PtrPoly,"%f %f",&Temp2,&Temp3);
      Vertices[++VerticesAmount].X =Temp2;
      Vertices[VerticesAmount].Y =Temp3;
      Vertices =realloc(Vertices,(VerticesAmount+2)*sizeof(Coordinates));
   }
```

```
}
fclose(PtrPoly);
}
void Dressing()
{
int     i,j,k,l,
        Number=0,
        Start=0;
int     *Flags;
printf("Dressing module ...\n");
if ((Flags=malloc(2*sizeof(int)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
if ((VerticesNumbered =calloc((VerticesAmount+2),sizeof(int)))==NULL) {
   perror("No memory ...\n");
   exit(0);
}
for (i=1;i<=VerticesAmount;i++) {
   Flags[i] =1;
   Flags =realloc(Flags,(i+2)*sizeof(int));
}
for (i=1;i<=VerticesAmount;i++) {
   if (Flags[i]==1) {
      ++Number;
      VerticesNumbered[i] =Number;
      for (j=(i+1);j<=VerticesAmount;j++) {
         if (Flags[j]==1) {
            if (Vertices[i].X==Vertices[j].X && Vertices[i].Y==Vertices[j].Y) {
               VerticesNumbered[j] =Number;
               Flags[j] =0;
            }
         }
      }
   }
}
free(Flags);
VerticesTrueAmount =Number;
if((VerticesSorted=calloc((VerticesTrueAmount+2),sizeof(Coordinates)))==NULL) {
   perror("No memory ...\n");
   exit(0);
}
for (i=1;i<=VerticesAmount;i++) {
   VerticesSorted[VerticesNumbered[i]].X =Vertices[i].X;
   VerticesSorted[VerticesNumbered[i]].Y =Vertices[i].Y;
}
if ((BondsExcess=malloc(2*sizeof(Ends)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
k =0;
for (i=1;i<=CellAmount;i++) {
   for (j=1;j<=(Cell[i]-1);j++) {
      BondsExcess[++k].End1 =VerticesNumbered[Start+j];
      BondsExcess[k].End2 =VerticesNumbered[Start+j+1];
      BondsExcess =realloc(BondsExcess,(k+2)*sizeof(Ends));
   }
   BondsExcess[++k].End1 =VerticesNumbered[Start+Cell[i]];
   BondsExcess[k].End2 =VerticesNumbered[Start+1];
   BondsExcess =realloc(BondsExcess,(k+2)*sizeof(Ends));
   Start =Start+Cell[i];
}
BondsExcessAmount =k;
for (i=1;i<=BondsExcessAmount;i++) {
   if (BondsExcess[i].End1 > BondsExcess[i].End2) {
      j =BondsExcess[i].End1;
      BondsExcess[i].End1 =BondsExcess[i].End2;
      BondsExcess[i].End2 =j;
```

```
   }
}
if ((Flags=malloc(2*sizeof(int)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
for (i=1;i<=BondsExcessAmount;i++) {
   Flags[i] =1;
   Flags =realloc(Flags,(i+2)*sizeof(int));
}
if ((Bonds=malloc(2*sizeof(Ends)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
k =0;
for (i=1;i<=BondsExcessAmount;i++) {
   for (j=(i+1);j<=BondsExcessAmount;j++) {
      if (BondsExcess[i].End1==BondsExcess[j].End1 && BondsExcess[i].End2==BondsExcess[j].End2)
      Flags[j] =0;
   }
}
k =0;
for (i=1;i<=BondsExcessAmount;i++) {
   if (Flags[i]==1) {
      Bonds[++k].End1 =BondsExcess[i].End1;
      Bonds[k].End2 =BondsExcess[i].End2;
      Bonds =realloc(Bonds,(k+2)*sizeof(Ends));
   }
}
BondsAmount =k;
free(Flags);
if ((Neighbours=malloc(2*sizeof(Near)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
k =0;
for (i=1;i<=BondsAmount;i++) {
   for (j=(i+1);j<=BondsAmount;j++) {
      if (Bonds[i].End1==Bonds[j].End1 || Bonds[i].End1==Bonds[j].End2 ||
      Bonds[i].End2==Bonds[j].End1 || Bonds[i].End2==Bonds[j].End2) {
         Neighbours[++k].Neighbour1 =i;
         Neighbours[k].Neighbour2 =j;
         Neighbours =realloc(Neighbours,(k+2)*sizeof(Near));
         Neighbours[++k].Neighbour1 =j;
         Neighbours[k].Neighbour2 =i;
         Neighbours =realloc(Neighbours,(k+2)*sizeof(Near));
      }
   }
}
NeighboursAmount =k;
if ((LeftRim=malloc(2*sizeof(int)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
if ((RightRim=malloc(2*sizeof(int)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
k =l =0;
for (i=1;i<=BondsAmount;i++) {
   if (VerticesSorted[Bonds[i].End1].X < LowerValueFrame ||
   VerticesSorted[Bonds[i].End2].X < LowerValueFrame) {
      LeftRim[++k] =i;
      LeftRim =realloc(LeftRim,(k+2)*sizeof(int));
   }
   if (VerticesSorted[Bonds[i].End1].X > HigherValueFrame ||
   VerticesSorted[Bonds[i].End2].X > HigherValueFrame) {
      RightRim[++l] =i;
```

```
      RightRim =realloc(RightRim,(l+2)*sizeof(int));
   }
}
LeftRimAmount =k;
RightRimAmount =l;
}
void PercolateCheck() {
int     i,j,k,l,m,n,
        Percolate=0,
        BlockedBond,
        ClustersAmount,
        TempClustersAmount,
        TouchedCondition,
        ClusterExist,
        Overlapped,
        TotalBlocked=0;
Group   *TempClusters,
        *Clusters;
FILE    *PtrVSorted,
        *PtrPercolC,
        *PtrMiscell;
printf("PercolateCheck module ...\n");
if ((TempClusters=malloc(2*sizeof(Group)))==NULL) {
   printf("No memory ...\n");
   exit(0);
}
TempClustersAmount =0;
srand(time(NULL));
while (Percolate==0) {
   BlockedBond =ceil((1.0*BondsAmount*(rand()%10001))/10000);
   free(Clusters);
   if ((Clusters=malloc(2*sizeof(Group)))==NULL) {
      printf("No memory 1...\n");
      exit(0);
   }
   ClustersAmount =0;
   Clusters[++ClustersAmount].MembersAmount =0;
   Clusters =realloc(Clusters,(ClustersAmount+2)*sizeof(Group));
   if ((Clusters[ClustersAmount].Members=malloc(2*sizeof(int)))==NULL) {
      printf("No memory ...\n");
      exit(0);
   }
   Clusters[ClustersAmount].Members[++(Clusters[ClustersAmount].MembersAmount)] =BlockedBond;
   Clusters[ClustersAmount].Members =realloc(Clusters[ClustersAmount].Members,
   (Clusters[ClustersAmount].MembersAmount+2)*sizeof(int));
   Overlapped =0;
   for (i=1;i<=TempClustersAmount;i++) {
      for (j=1;j<=TempClusters[i].MembersAmount;j++) {
         if (BlockedBond==TempClusters[i].Members[j]) {
            Overlapped =1;
            break;
         }
      }
      if (Overlapped==1)
         break;
   }
   if (Overlapped==0) {
      ++TotalBlocked;
      if (TotalBlocked > (BondsAmount -2)) {
         printf("Too far ...\n");
         exit(0);
      };
      for (i=1;i<=TempClustersAmount;i++) {
         TouchedCondition =0;
         for (j=1;j<=NeighboursAmount;j++) {
            for (k=1;k<=TempClusters[i].MembersAmount;k++) {
               if (Neighbours[j].Neighbour1==BlockedBond &&
               Neighbours[j].Neighbour2==TempClusters[i].Members[k]) {
```

```
            TouchedCondition =1;
            break;
         };
      }
      if (TouchedCondition==1)
         break;
   }
   ClusterExist =0;
   for (j=1;j<=TempClusters[i].MembersAmount;j++) {
      ClusterExist +=TempClusters[i].Members[j];
   }
   if (TouchedCondition==0 && ClusterExist!=0) {
      if ((Clusters[++ClustersAmount].Members=malloc(2*sizeof(int)))==NULL) {
         printf("No memory ...\n");
         exit(0);
      };
      Clusters =realloc(Clusters,(ClustersAmount+2)*sizeof(Group));
      Clusters[ClustersAmount].MembersAmount =0;
      for (j=1;j<=TempClusters[i].MembersAmount;j++) {
         Clusters[ClustersAmount].Members[++Clusters[ClustersAmount].MembersAmount] =
         TempClusters[i].Members[j];
         Clusters[ClustersAmount].Members =realloc(Clusters[ClustersAmount].Members,
         (Clusters[ClustersAmount].MembersAmount+2)*sizeof(int));
      }
   }
   else {
      for (j=1;j<=TempClusters[i].MembersAmount;j++) {
         Clusters[1].Members[++(Clusters[1].MembersAmount)] =TempClusters[i].Members[j];
         Clusters[1].Members =realloc(Clusters[1].Members,
         (Clusters[1].MembersAmount+2)*sizeof(int));
      }
   };
}
for (i=1;i<=ClustersAmount;i++) {
   for (j=1;j<=Clusters[i].MembersAmount;j++) {
      for (k=1;k<=LeftRimAmount;k++) {
         for (l=1;l<=RightRimAmount;l++) {
            for (m=1;m<=Clusters[i].MembersAmount;m++) {
               if (Clusters[i].Members[j]==LeftRim[k] &&
               Clusters[i].Members[m]==RightRim[l]) {
                  Percolate =1;
                  PtrVSorted =fopen("./VSorted","w");
                  fprintf(PtrVSorted,"%d\n",VerticesTrueAmount);
                  for (n=1;n<=VerticesTrueAmount;n++) {
                     fprintf(PtrVSorted,"%5.5f %5.5f\n",VerticesSorted[n].X,
                     VerticesSorted[n].Y);
                  }
                  fclose(PtrVSorted);
                  PtrPercolC =fopen("./PercolC","w");
                  fprintf(PtrPercolC,"%d\n",Clusters[i].MembersAmount);
                  for (n=1;n<=Clusters[i].MembersAmount;n++) {
                     fprintf(PtrPercolC,"%d %d\n",Bonds[Clusters[i].Members[n]].End1,
                     Bonds[Clusters[i].Members[n]].End2);
                  }
                  fclose(PtrPercolC);
                  PtrMiscell =fopen("./Miscell","w");
                  fprintf(PtrMiscell,"%d\n%d\n%d\n%d\n%3.3f\n",CellAmount,BondsAmount,
                  Bonds[BlockedBond].End1,Bonds[BlockedBond].End2,
                  ((1.0*TotalBlocked)/BondsAmount));
                  fclose(PtrMiscell);
                  printf("End ...\n");
                  exit(0);
               }
            }
         }
      }
   }
}
```

```
      free(TempClusters);
      if ((TempClusters=malloc(2*sizeof(Group)))==NULL) {
         printf("No memory ...\n");
         exit(0);
      }
      TempClustersAmount =0;
      for (i=1;i<=ClustersAmount;i++) {
         TempClusters[++TempClustersAmount].MembersAmount =0;
         if ((TempClusters[TempClustersAmount].Members=malloc(2*sizeof(int)))==NULL) {
            printf("No memory ...\n");
            exit(0);
         }
         TempClusters =realloc(TempClusters,(TempClustersAmount+2)*sizeof(Group));
         for (j=1;j<=Clusters[i].MembersAmount;j++) {
            TempClusters[TempClustersAmount].
            Members[++(TempClusters[TempClustersAmount].MembersAmount)]
            = Clusters[i].Members[j];
            TempClusters[TempClustersAmount].Members =
            realloc(TempClusters[TempClustersAmount].Members,
            (TempClusters[TempClustersAmount].MembersAmount+2)*sizeof(int));
         }
      }
   }
}
free(TempClusters);
free(Clusters);
}
main()
{
ReadFiles();
Dressing();
PercolateCheck();
free(VerticesNumbered);
free(Vertices);
free(Cell);
free(Nuclei);
free(VerticesSorted);
free(BondsExcess);
free(Bonds);
free(Neighbours);
free(LeftRim);
free(RightRim);
}
```

B.2.2.2 Program written from a control area algorithm

This program follows the Algorithm 4. It is used for producing Figures 4.26, 4.27, 4.28, 4.29, 4.30, 4.34, 4.32, 4.33, 4.34.

```
clear;                    clf;               echo on;
LowerValueFrame =0.1;     HigherValueFrame =0.9;
PtrPPyIdx =fopen('./PPyIdx300L03','r');
Temp =fscanf(PtrPPyIdx,'%d %d');
for i=1:2:(size(Temp)-1),
   CellMap =[CellMap;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrPPyIdx);
CellAmount =size(CellMap,1);
PtrP =fopen('./P300L03','r');
TempAmount =fscanf(PtrP,'%d',1);
Temp =fscanf(PtrP,'%f %f',inf);
for i=1:2:(size(Temp)-1),
   Nuclei =[Nuclei;Temp(i,:),Temp((i+1),:)];
end
fclose(PtrP);
NucleiAmount =size(Nuclei,1);
PtrPoly =fopen('./Poly300L03','r');
i =0;
while i<CellAmount,
```

```
   i =i+1;
   TempAmount =fscanf(PtrPoly,'%d',1);
   Cell =[Cell;TempAmount];
   Temp =fscanf(PtrPoly,'%f %f',(2*TempAmount));
   for j=1:2:(size(Temp)-1),
      Vertices =[Vertices;Temp(j,:),Temp((j+1),:)];
   end
end
fclose(PtrPoly);
VerticeAmount =size(Vertices,1);
Flags =ones(VerticeAmount,1);
VerticeNumbered =zeros(VerticeAmount,1);
Number =0;
for i=1:VerticeAmount,
   if Flags(i,1)==1,
      Number =Number+1;
      VerticeNumbered(i,1) =Number;
      for j=(i+1):VerticeAmount,
         if Flags(j,1)==1,
            if (Vertices(i,:)-Vertices(j,:))==[0,0],
               VerticeNumbered(j,1) =Number;
               Flags(j,1) =0;
            end
         end
      end
   end
end
VerticeTrueAmount =max(VerticeNumbered);
VerticesSorted =zeros(VerticeTrueAmount,2);
for i=1:VerticeAmount,
   VerticesSorted(VerticeNumbered(i,1),:) =Vertices(i,:);
end
Start =0
for i=1:CellAmount,
   for j=1:(Cell(i,1)-1),
      BondsExcess =[BondsExcess;VerticeNumbered((Start+j),1),VerticeNumbered((Start+j+1),1)];
   end
   BondsExcess =[BondsExcess;VerticeNumbered((Start+Cell(i,1)),1),VerticeNumbered((Start+1),1)];
   Start =Start+Cell(i,1);
end
BondsExcess =sort(BondsExcess')';
Flags =ones(size(BondsExcess,1),1);
for i=1:size(BondsExcess,1),
   for j=(i+1):size(BondsExcess,1),
      if (BondsExcess(i,:)-BondsExcess(j,:))==[0, 0],
         Flags(j,1) =0;
      end
   end
end
for i=1:size(BondsExcess,1),
   if Flags(i,1)==1
      Bonds =[Bonds;BondsExcess(i,:)];
   end
end
BondsAmount =size(Bonds,1);
Neighbours =sparse([],[],[],size(Bonds),size(Bonds),0);
for i=1:BondsAmount,
   for j=(i+1):size(Bonds,1),
      if ((Bonds(i,1)==Bonds(j,1)) |(Bonds(i,1)==Bonds(j,2)))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      elseif ((Bonds(i,2)==Bonds(j,1)) |(Bonds(i,2)==Bonds(j,2)))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      end
   end
end
LeftRim =sparse([],[],[],size(Bonds),1,0);
```

```
RightRim =sparse([],[],[],size(Bonds),1,0);
for i=1:BondsAmount,
   if ((VerticesSorted(Bonds(i,1),1) < LowerValueFrame) | ...
   (VerticesSorted(Bonds(i,2),1) < LowerValueFrame))
      LeftRim(i,1) =1;
   end
   if ((VerticesSorted(Bonds(i,1),1) > HigherValueFrame) | ...
   (VerticesSorted(Bonds(i,2),1) > HigherValueFrame))
      RightRim(i,1) =1;
   end
end
clear Flags;
Flags =sparse([],[],[],BondsAmount,1,0);
for i=1:BondsAmount,
   if ((((VerticesSorted(Bonds(i,1),1) > LowerValueFrame) &...
   (VerticesSorted(Bonds(i,1),1) < HigherValueFrame)) &...
   ((VerticesSorted(Bonds(i,1),2) > LowerValueFrame) &...
   (VerticesSorted(Bonds(i,1),2) < HigherValueFrame))) | ...
   (((VerticesSorted(Bonds(i,2),1) > LowerValueFrame) &...
   (VerticesSorted(Bonds(i,2),1) < HigherValueFrame)) &...
   ((VerticesSorted(Bonds(i,2),2) > LowerValueFrame) &...
   (VerticesSorted(Bonds(i,2),2) < HigherValueFrame))))
      Flags(i,1) =1;
   end
end
Total =sum(full(Flags));
TempClusters =sparse([],[],[],BondsAmount,1,0)
Percolate =0;
while (sum(sum(TempClusters)) < Total),
   BlockedBond =ceil(BondsAmount*rand(1));
   while (Flags(BlockedBond,1)~=1),
      BlockedBond =ceil(BondsAmount*rand(1));
   end
   BlockedMask =sparse([BlockedBond],[1],[1],BondsAmount,1,1);
   clear Clusters;
   Clusters =BlockedMask;
   SizeTempClusters =size(TempClusters,2);
   Overlapped =0;
   for i=1:SizeTempClusters,
      Overlapped =Overlapped + max(BlockedMask & TempClusters(:,i));
   end
   clear BlockedMask;
   if Overlapped==0
      for i=1:SizeTempClusters,
         TouchCondition =max(Neighbours(:,BlockedBond) & TempClusters(:,i));
         ClusterExist =max(TempClusters(:,i));
         if ((TouchCondition==0) & (ClusterExist~=0))
            Clusters =[Clusters,TempClusters(:,i)];
         else
            Clusters(:,1) =Clusters(:,1) | TempClusters(:,i);
         end
      end
      i =0;
      ClustersAmount =size(Clusters,2);
      SumClusters =sum(Clusters);
      TempNumberClusters =sparse([],[],[],BondsAmount,1,0);
      if Percolate==0
         while (i < ClustersAmount),
            i =i+1;
            if ((max(Clusters(:,i) & LeftRim) > 0) & (max(Clusters(:,i) & RightRim) > 0))
               Percolate =1;
               PercolatingCluster =Clusters(:,i);
               ClusterForPlot =Clusters;
               KeyBond =BlockedBond;
               subplot(2,2,1);
               hold on;
               plot(Nuclei(:,1),Nuclei(:,2),'k.');
               Count = 1;
```

```
            Loop = size(Cell);
            for i=1:Loop;
        SmallLoop = Cell(i,1);
        for j=Count:(Count+SmallLoop-2);
                  x = [Vertices(j,1) Vertices(j+1,1)];
                  y = [Vertices(j,2) Vertices(j+1,2)];
                  plot(x,y,'c');
        end
               x = [Vertices(Count,1) Vertices(j+1,1)];
               y = [Vertices(Count,2) Vertices(j+1,2)];
               plot(x,y,'c');
               Count = Count + SmallLoop;
            end
            plot([0 1 1],[1 1 0],'k');
            [IClusterForPlot,JClusterForPlot] =find(ClusterForPlot);
            for i=1:size(IClusterForPlot,1),
               plot([VerticesSorted(Bonds(IClusterForPlot(i),1),1),...
               VerticesSorted(Bonds(IClusterForPlot(i),2),1)],...
               [VerticesSorted(Bonds(IClusterForPlot(i),1),2),...
               VerticesSorted(Bonds(IClusterForPlot(i),2),2)],'r');
            end
            plot([LowerValueFrame HigherValueFrame HigherValueFrame LowerValueFrame ...
            LowerValueFrame],...
            [HigherValueFrame HigherValueFrame LowerValueFrame LowerValueFrame ...
            HigherValueFrame],'b');
            title('Blockages at the percolation');
            subplot(2,2,2);
            hold on;
            plot(Nuclei(:,1),Nuclei(:,2),'k.');
            Count = 1;
            Loop = size(Cell);
            for i=1:Loop;
               SmallLoop = Cell(i,1);
               for j=Count:(Count+SmallLoop-2);
                  x = [Vertices(j,1) Vertices(j+1,1)];
                  y = [Vertices(j,2) Vertices(j+1,2)];
                  plot(x,y,'c');
               end
               x = [Vertices(Count,1) Vertices(j+1,1)];
               y = [Vertices(Count,2) Vertices(j+1,2)];
               plot(x,y,'c');
               Count = Count + SmallLoop;
            end
            plot([0 1 1],[1 1 0],'k');
            [IPercolatingCluster,JPercolatingCluster] =find(PercolatingCluster);
            for i=1:size(IPercolatingCluster,1),
               plot([VerticesSorted(Bonds(IPercolatingCluster(i),1),1),...
               VerticesSorted(Bonds(IPercolatingCluster(i),2),1)],...
               [VerticesSorted(Bonds(IPercolatingCluster(i),1),2),...
               VerticesSorted(Bonds(IPercolatingCluster(i),2),2)],'r');
            end
            plot([VerticesSorted(Bonds(KeyBond,1),1),VerticesSorted(Bonds(KeyBond,2),1)],...
            [VerticesSorted(Bonds(KeyBond,1),2),VerticesSorted(Bonds(KeyBond,2),2)],'b');
            plot([LowerValueFrame HigherValueFrame HigherValueFrame ...
            LowerValueFrame LowerValueFrame],...
            [HigherValueFrame HigherValueFrame LowerValueFrame LowerValueFrame ...
            HigherValueFrame],'b');
            title('The bond that causes percolation');
            TotalBlocked =sum(sum(ClusterForPlot));
            CriticalProb =TotalBlocked/Total;
            clear ClusterForPlot;
            clear PercolatingCluster;
         end
      end
   end
   i =0;
   while i < ClustersAmount,
      i =i+1;
```

```
         TempNumberClusters(SumClusters(1,i),1) =TempNumberClusters(SumClusters(1,i),1)+1;
      end
      if (max(SumClusters)<(BondsAmount/2))
         ForPcCurve =[ForPcCurve;(sum(SumClusters)/BondsAmount), (max(SumClusters))];
      end
      TempClusters =Clusters;
      NumberClusters =[NumberClusters,TempNumberClusters];
   end
end
subplot(2,2,4)
hold on
ForPcCurve =full(ForPcCurve*([(BondsAmount/Total),0;0,1]));
plot(ForPcCurve(:,1),ForPcCurve(:,2),'k');
axis ([0 1.0 0 Total]);
grid on;
title('n vs p');
xlabel('p');
ylabel('n(i)max');
axis([0 1.0 0 200]);
for i=1:BondsAmount,
   ClustersSizeFactor(i,1) =i;
end
SumNumberClusters =sum(NumberClusters.*(ClustersSizeFactor*ones(1,size(NumberClusters,2))));
TempSize =size(find(SumNumberClusters),1);
XN =SumNumberClusters(1,1:TempSize)/BondsAmount;
YN1 =NumberClusters(1,1:TempSize)./SumNumberClusters(1,1:TempSize);
YN2 =(2*NumberClusters(2,1:TempSize))./SumNumberClusters(1,1:TempSize);
YN3 =(3*NumberClusters(3,1:TempSize))./SumNumberClusters(1,1:TempSize);
YN20 =(20*NumberClusters(20,1:TempSize))./SumNumberClusters(1,1:TempSize);
NiForPlot =[XN;YN1;YN2;YN3;YN20];
subplot(2,2,3);
hold on;
NiForPlot =full(NiForPlot);
plot(NiForPlot(1,:),NiForPlot(2,:),'b');
plot(NiForPlot(1,:),NiForPlot(3,:),'g');
plot(NiForPlot(1,:),NiForPlot(4,:),'m');
plot(NiForPlot(1,:),NiForPlot(5,:),'k');
plot([0 1 1],[1 1 0],'k');
grid on;
title('n1, n2, n3 and n20 vs p');
xlabel('p');
ylabel('n(i)/n');
```

Appendix C

Generating uniform lattices

C.1 Square lattice

A square lattice has got coordination number of four. The following algorithm is used in generating such lattice on MATLAB.

```
XRange =60;
YRange =60;
for j=1:(YRange+1),
   for i=1:(XRange+1),
      Vertices =[Vertices;i,j];
   end
end
VerticesAmount =size(Vertices,1);
for j=1:(YRange+1),
   for i=1:XRange,
      BondsXY =[BondsXY;i,j,(i+1),j];
   end
end
for i=1:(XRange+1),
   for j=1:YRange,
      BondsXY =[BondsXY;i,j,i,(j+1)];
   end
end
BondsAmount =size(BondsXY,1);
Bonds =zeros(BondsXY,2);
for i=1:BondsAmount,
   for j=1:VerticesAmount,
      if (([BondsXY(i,1),BondsXY(i,2)]-Vertices(j,:))==[0,0])
         Bonds(i,1) =j;
      elseif (([BondsXY(i,3),BondsXY(i,4)]-Vertices(j,:))==[0,0])
         Bonds(i,2) =j;
      end
  end
end
Neighbours =sparse([],[],[],size(Bonds),size(Bonds),0);
for i=1:BondsAmount,
   for j=(i+1):size(Bonds,1),
      if (Bonds(i,1)==Bonds(j,1)) |(Bonds(i,1)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      elseif (Bonds(i,2)==Bonds(j,1)) |(Bonds(i,2)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      end
   end
end
```

C.2 Triangular lattice

A triangular lattice has got a coordination number of six. The following codes in MATLAB is used to generate such a lattice.

```
XRange =60;
YRange =60;
Count =0;
Rows =0;
for j=1:sqrt(3):YRange,
   Rows =Rows+1;
   for i=1:XRange,
      Count =Count+1;
      Vertices =[Vertices;i,j];
      Bonds =[Bonds;Count,(Count+1)];
   end
   Count =Count+1;
   Vertices =[Vertices;(i+1),j];
end
Between =Count;
RowBetween =0;
j =1+(sqrt(3)/2);
while (RowBetween<=(Rows-2)),
   RowBetween =RowBetween+1;
   for i=1.5:(XRange-.5),
      Count =Count+1;
      Vertices =[Vertices;i,j];
      Bonds =[Bonds;Count,(Count+1)];
   end
   Count =Count+1;
   Vertices =[Vertices;(i+1),j];
   j =j+sqrt(3);
end
TotalVertices =Count;
Start =1;
for i=0:(Rows-2),
   for j=(Between+(i*XRange)+1):(Between+((i+1)*XRange)),
      Bonds =[Bonds;j,Start];
      Bonds =[Bonds;j,(Start+1)];
      Bonds =[Bonds;j,(Start+XRange+1)];
      Bonds =[Bonds;j,(Start+XRange+2)];
      Start =Start+1;
   end
   Start =Start+1;
end
BondsAmount =size(Bonds,1);
Neighbours =sparse([],[],[],size(Bonds),size(Bonds),0);
for i=1:BondsAmount,
   for j=(i+1):size(Bonds,1),
      if (Bonds(i,1)==Bonds(j,1)) |(Bonds(i,1)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      elseif (Bonds(i,2)==Bonds(j,1)) |(Bonds(i,2)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      end
   end
end
```

C.3 Honeycomb lattice

A honeycomb lattice has got a coordination number of three. It can be produced by taking away one bond from each node in a square lattice. The following code in MATLAB is for producing such a lattice.

```
XRange =20;
YRange =20;
Y =1;
Count =0;
```

```
Rows =0;
for j=1:3:(YRange-3),
   Rows =Rows+1;
   for i=1:sqrt(3):(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,Y];
   end
   if (j==1)
      StartRow2 =Count+1;
   end
   Y =Y+.5;
   for i=(1-(.5*sqrt(3))):sqrt(3):(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,Y];
   end
   if (j==1)
      StartRow3 =Count+1;
   end
   Y =Y+1;
   for i=(1-(.5*sqrt(3))):sqrt(3):(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,Y];
   end
   if (j==1)
      StartRow4 =Count+1;
   end
   Y =Y+.5;
   for i=1:sqrt(3):(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,Y];
   end
   Y =Y+1;
   if (j==1)
      Period =Count;
   end
end
Y =Y-.5;
for i=(1-(.5*sqrt(3))):sqrt(3):(XRange+1),
   Vertices =[Vertices;i,Y];
end
for j=0:(Rows-1),
   for i=(j*Period+1):(j*Period+StartRow2-1),
      Bonds =[Bonds;i,(i+StartRow2-1)];
      Bonds =[Bonds;i,(i+StartRow2)];
   end
   for i=(j*Period+StartRow2):(j*Period+StartRow3-1),
      Bonds =[Bonds;i,(i+StartRow3-StartRow2)];
   end
   for i=(j*Period+StartRow3):(j*Period+StartRow4-2),
      Bonds =[Bonds;i,(i+StartRow4-StartRow3)];
      Bonds =[Bonds;(i+1),(i+StartRow4-StartRow3)];
   end
   for i=(j*Period+StartRow4):((j+1)*Period),
      Bonds =[Bonds;i,(i+Period-StartRow4+1)];
   end
end
VerticesAmount =size(Vertices,1);
BondsAmount =size(Bonds,1);
Neighbours =sparse([],[],[],size(Bonds),size(Bonds),0);
for i=1:BondsAmount,
   for j=(i+1):size(Bonds,1),
      if (Bonds(i,1)==Bonds(j,1)) |(Bonds(i,1)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      elseif (Bonds(i,2)==Bonds(j,1)) |(Bonds(i,2)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      end
```

```
   end
end
```

C.4 Kagomé lattice

A Kagomé lattice has got a coordination number of four. It can be produced by taking away at least two bonds from each node in a triangular lattice. This can be shown in Figure C.1. The following MATLAB

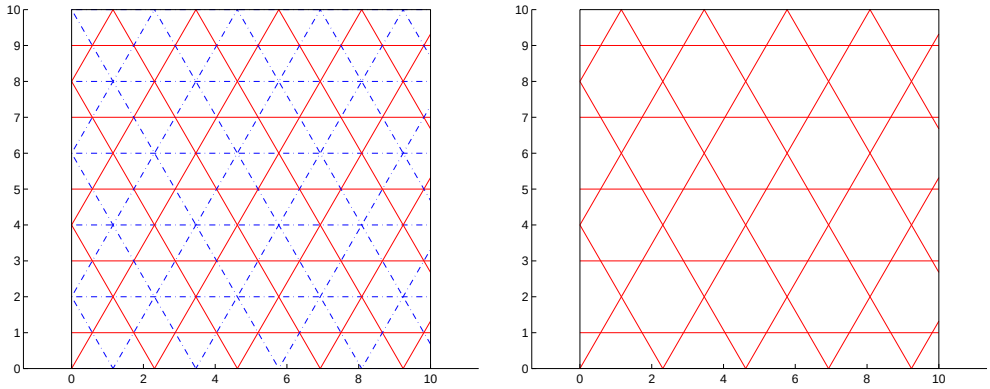


Figure C.1: *Producing a Kagomé lattice from a triangular lattice*

code is used to produce a Kagomé lattice for bond percolation.

```
XRange =60;
YRange =60;
Count =0;
Rows =0;
for j=1:(2*sqrt(3)):YRange,
   Rows =Rows+1;
   for i=1:2:(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,j];
   end
   if j==1
      StartRow2 =Count+1;
   end
   Y =j+(0.5*sqrt(3));
   for i=1.5:(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,Y];
      Bonds =[Bonds;Count,(Count+1)];
   end
   Bonds =Bonds(1:(size(Bonds,1)-1),:);
   if j==1
      StartRow3 =Count+1;
   end
   Y =j+sqrt(3);
   for i=2:2:(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,Y];
   end
   if j==1
      StartRow4 =Count+1;
   end
   Y =j+(1.5*sqrt(3));
   for i=1.5:(XRange+1),
      Count =Count+1;
      Vertices =[Vertices;i,Y];
```

```
      Bonds =[Bonds;Count,(Count+1)];
   end
   Bonds =Bonds(1:(size(Bonds,1)-1),:);
   if j==1
      Period =Count;
   end
end
Y =Y+(0.5*sqrt(3));
for i=1:2:(XRange+1),
   Count =Count+1;
   Vertices =[Vertices;i,Y];
end
VerticesAmount =size(Vertices,1);
for i=0:(Rows-1),
   Start1 =i*Period+1;
   Start2 =i*Period+StartRow2;
   Start4 =i*Period+StartRow4;
   for j=(i*Period+StartRow3):(i*Period+StartRow3+XRange/2-1),
      Bonds =[Bonds;j,Start2];
      Bonds =[Bonds;j,(Start2+1)];
      Bonds =[Bonds;j,Start4];
      Bonds =[Bonds;j,(Start4+1)];
      Bonds =[Bonds;Start2,Start1];
      Bonds =[Bonds;(Start2+1),(Start1+1)];
      Bonds =[Bonds;Start4,(Start1+Period)];
      Bonds =[Bonds;(Start4+1),(Start1+Period+1)];
      Start1 =Start1+1;
      Start2 =Start2+2;
      Start4 =Start4+2;
   end
end
BondsAmount =size(Bonds,1);
Neighbours =sparse([],[],[],size(Bonds),size(Bonds),0);
for i=1:BondsAmount,
   for j=(i+1):size(Bonds,1),
      if (Bonds(i,1)==Bonds(j,1)) |(Bonds(i,1)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      elseif (Bonds(i,2)==Bonds(j,1)) |(Bonds(i,2)==Bonds(j,2))
         Neighbours(i,j) =1;
         Neighbours(j,i) =1;
      end
   end
end
```

References

[1] N. Ahuja. Dot pattern processing using Voronoi neighborhoods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(3):336–343, May 1982.

[2] N. Ahuja and B. J. Schachter. *Pattern models*. John Wiley & Sons Inc., 1983.

[3] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, September 1991.

[4] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984.

[5] D. J. Bell, P. Deckmyn, and G. A. Davies. Percolating clusters on Voronoi lattices and the relationship to particle fouling on filters. In *Proceedings of ICIAM 95*, Hamburg, July 1995. ICIAM.

[6] K. Binder and D. Stauffer. *Monte Carlo studies of "random" systems*, volume 36 of *Topics in current physics*, chapter 8, pages 241–275. Springer-Verlag, 1984.

[7] A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[8] S. R. Broadbent and J. M. Hammersley. Percolation processes I. crystals and mazes. *Camb. Phil. Soc.*, 53(3):629–641, 1957.

[9] W. Brostow, J.-P. Dussault, and B. L. Fox. Construction of Voronoi polyhydra. *Journal of Computational Physics*, 29:81–92, 1978.

[10] J. P. Clerc, G. Giraud, J. Roussenq, R. Blanc, J. P. Carton, E. Guyon, H. Ottavi, and D. Stauffer. La percolation: modèles, simulations analogiques et numériques. *Annales de Physique*, 8(1):5–105, 1983.

[11] R. Cole, M. T. Goodrich, and C. Ó. Dúnlaing. Merging free trees in parallel for efficient Voronoi diagram construction. In M. S. Paterson, editor, *Lecture Notes in Computer Science*, pages 432–445. Springer-Verlag, 1990.

[12] A. Coniglio. Fractal structure of Ising and Potts clusters: static and dynamic approach. In L. Pietronero, editor, *Fractals' physical origin and properties*, pages 123–133. Plenum Press, New York, 1989.

[13] G. A. Davies and D. J. Bell. Voronoi tessellations to describe pattern structures in science and technology. In *Proceedings of ICIAM 95*, Hamburg, July 1995. ICIAM.

[14] P. Deckmyn, G. A. Davies, and D. J. Bell. Properties of percolating clusters on finite lattices applied to model filtration processes. *Applied Mathematical Modelling*, 19:258–269, 1995.

[15] G. Deutscher, A. Kapitulnik, and M. Rappaport. Percolation in metal-insulator systems. In G. Deutscher, R. Zallen, and J. Adler, editors, *Annals of the Israel Physical Society*, volume 5, chapter 10, pages 207–227. Israel Physical Society, 1983.

[16] F. A. L. Dullien. *Porous media fluid transport and pore structure.* Academic Press, London, 1979.

[17] F. Fankhauser I, F. Fankhauser II, and H. Giger. A cluster and scotoma analysis based on empiric criteria. *Graefe's Archive for clinical and experimental Ophthalmology*, 231:697–703, 1993.

[18] M. E. Fisher and J. W. Essam. Some cluster size and percolation problems. *Journal of Mathematical Physics*, 2(4):609–619, July-August 1961.

[19] D. P. Fraser, M. J. Zuckermann, and O. G. Mouritsen. Simulation technique for hard-disk models in two dimensions. *The American Physical Society*, 42(6):3186–3195, September 1990.

[20] J.-J. Fu and R. C. T. Lee. Voronoi diagrams of moving points in the plane. In K. V. Nori and C. E. V. Madhavan, editors, *Lecture notes in computer science*, pages 238–254. Springer-Verlag, 1990.

[21] P. J. Green and R. Sibson. Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21(2):168–173, 1977.

[22] G. S. Grest and M. H. Cohen. Percolation and the glass transition. In G. Deutscher, R. Zallen, and J. Adler, editors, *Annals of the Israel Physical Society*, volume 5, chapter 9, pages 187–205. Israel Physical Society, 1983.

[23] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. In M. S. Paterson, editor, *Lecture Notes in Computer Science*, pages 414–431. Springer-Verlag, 1990.

[24] V. Icke and R. van de Weygaert. Fragmenting the universe, i. Statistics of two-dimensional Voronoi foams. *Astronomy and Astrophysics*, pages 16–32, 1987.

[25] N. M. Jackson. *A mathematical model to simulate the structure and performance of porous media*. PhD thesis, University of Manchester Institute of Science and Technology, Manchester, U.K., 1994.

[26] N. M. Jackson. A model to simulate the structure and performance of cellular polymeric membranes: structure, flux and filtration characteristics. *Filtration and Separation*, 1994.

[27] R. Jafferali. *A stochastic model to simulate the structure and performance of asymmetric porous media and the growth of animal cell cultures*. PhD thesis, University of Manchester Institute of Science and Technology, 1995.

[28] R. Jafferali, N. M. Jackson, and G. A. Davies. Analysis of the structure of porous polymeric membranes. In *Proceedings of ICIAM 95*, Hamburg, July 1995. ICIAM.

[29] B. Jouhier, C. Allain, B. Gauthier-Manuel, and E. Guyon. The sol-gel transition. In G. Deutscher, R. Zallen, and J. Adler, editors, *Annals of the Israel Physical Society*, volume 5, chapter 8, pages 167–185. Israel Physical Society, 1983.

[30] R. Klein. Concrete and abstract Voronoi diagrams. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[31] J. H. F. Lim. *Tessellations in 2 space to simulate cellular structures: application to membrane technology*. PhD thesis, University of Manchester Institute of Science and Technology, 1990.

[32] W.-M. Lu, K.-J. Hwang, and S.-C. Ju. Studies on the mechanism of cross-flow filtration. *Chemical Engineering Science*, 48(5):863–872, 1993.

[33] P. Meakin and A. T. Skjeltorp. Application of experimental and numerical models to the physics of multiparticle systems. *Advances in Physics*, 42(1):1–127, 1993.

[34] J. B. Parse and J. A. Wert. A geometrical description of particle distributions in materials. *Modelling & Simulation in Materials Science & Engineering*, 1(3):275–296, 1993.

[35] L. Pietronero, A. Erzan, and C. Evertsz. New theoretical methods for fractal growth. In L. Pietronero, editor, *Fractals' physical origin and properties*, pages 169–191. Plenum Press, New York, 1989.

[36] P. Pradános, J. I. Arribas, and A. Hernández. Retention of proteins in cross-flow UF through asymmetric inorganic membranes. *AIChE Journal*, 40(11):1901–1910, November 1994.

[37] V. G. J. Rodgers and R. E. Sparks. Reduction of membrane fouling in the ultrafiltration of binary protein mixtures. *AIChE Journal*, 37(10):1517–1528, October 1991.

[38] A.-C. Roudot, F. Duprat, and E. Pietri. Simulation of a penetrometric test on apples using Voronoi-Delaunay tessellation. *Food Structure*, 9:215–222, 1990.

[39] W. S. Russell and J. William W. Roberts. Analysis of the distribution of pitch angles in model galactic disks: numerical methods and algorithms. *The Astrophysical Journal*, 414:86–97, September 1993.

[40] A. E. Scheidegger. *The physics of flow through porous media*. University of Toronto Press, 1960.

[41] L. S. Schulman. Propagating stochastic star formation and galactic structure. In G. Deutscher, R. Zallen, and J. Adler, editors, *Annals of the Israel Physical Society*, volume 5, chapter 12, pages 251–296. Israel Physical Society, 1983.

[42] V. K. S. Shante and S. Kirkpatrick. An introduction to percolation theory. *Adv. Phys.*, 20:325–357, 1971.

[43] N. E. Sherman and J. D. Sherwood. Cross-flow filtration: cakes with variable resistance and capture efficiency. *Chemical Engineering Science*, 48(16):2913–2918, 1993.

[44] M. Sopek, W. M. Bartczak, and J. Kroh. Extended Molecular Dynamics study of molten mixtures LiBr/KBr. *Bulletin of the Polish Academy of Sciences Chemistry*, 41(2):101–120, 1993.

[45] D. Stauffer. *Introduction to percolation theory*. Taylor & Francis, London, 1985.

[46] S. Stifter. An axiomatic approach to Voronoi-diagrams in 3D. *Journal of Computer and System Sciences*, 43:361–379, 1991.

[47] A. Suzuki and M. Iri. Approximation of a tessellation of the plane by a Voronoi diagram. *Journal of the Operations Research*, 29(1):69–96, March 1986.

[48] V. Trimble. A large-scale puzzle. *Nature*, 345:665–666, June 1990.

[49] V. J. D. Tsai. Fast topological construction of Delaunay triangulations and Voronoi diagrams. *Computers & Geosciences*, 19(10):1463–1474, 1993.

[50] R. van de Weygaert. Fragmenting the universe III. the construction and statistics of 3-D Voronoi tessellations. *Astronomy and Astrophysics*, 283:361–406, 1994.

[51] R. van de Weygaert and V. Icke. Fragmenting the universe, II. Voronoi vertices as Abell clusters. *Astronomy and Astrophysics*, 213:1–9, 1989.

[52] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[53] E. T. Wilkinson and G. A. Davies. The application of percolation theory to the analysis of sieve blinding. *Chemical Engineering Science*, 44:459–470, 1989.